**Project4 Verilog完成单周期CPU开发**

**一、总体设计**

**1、顶层模块**

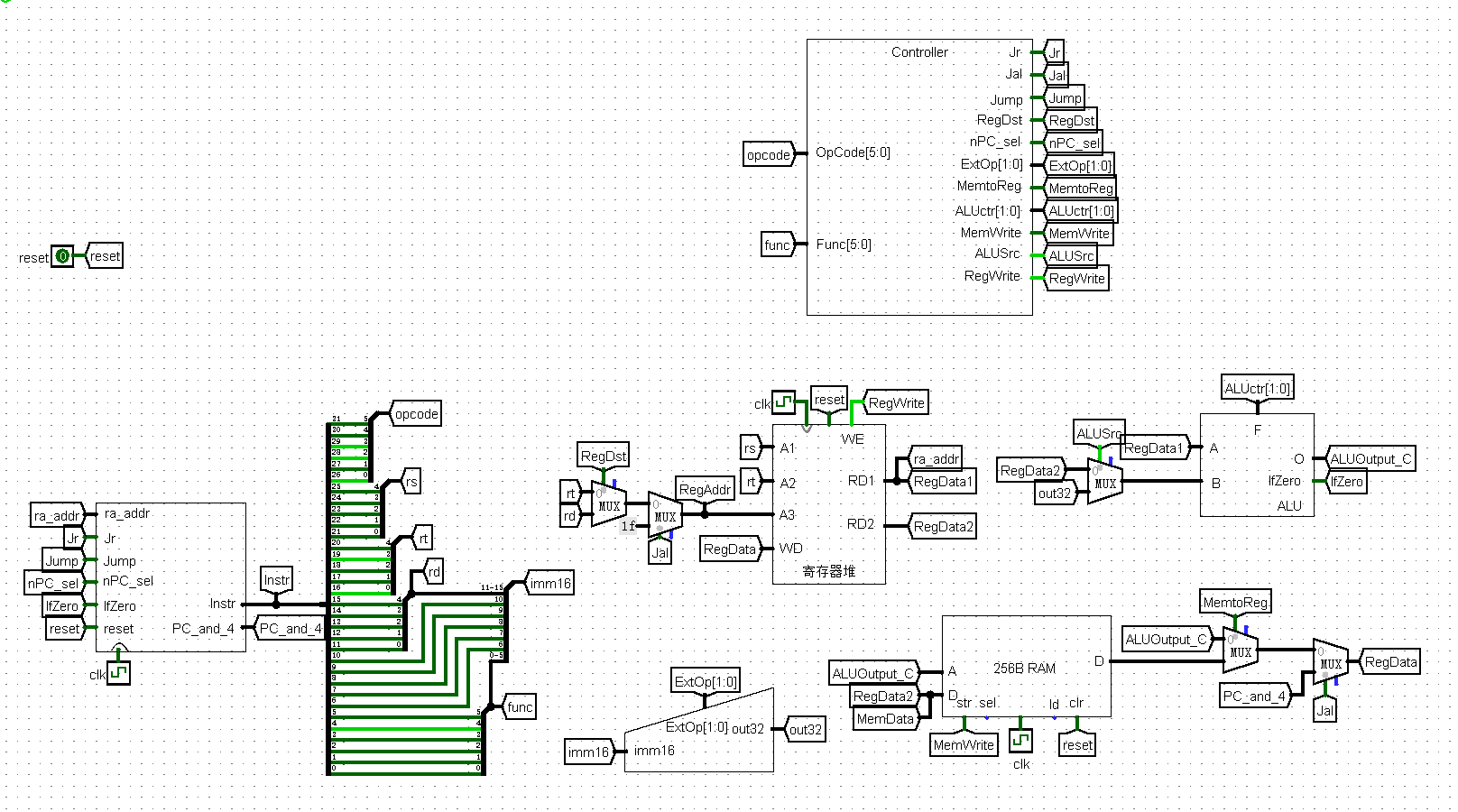
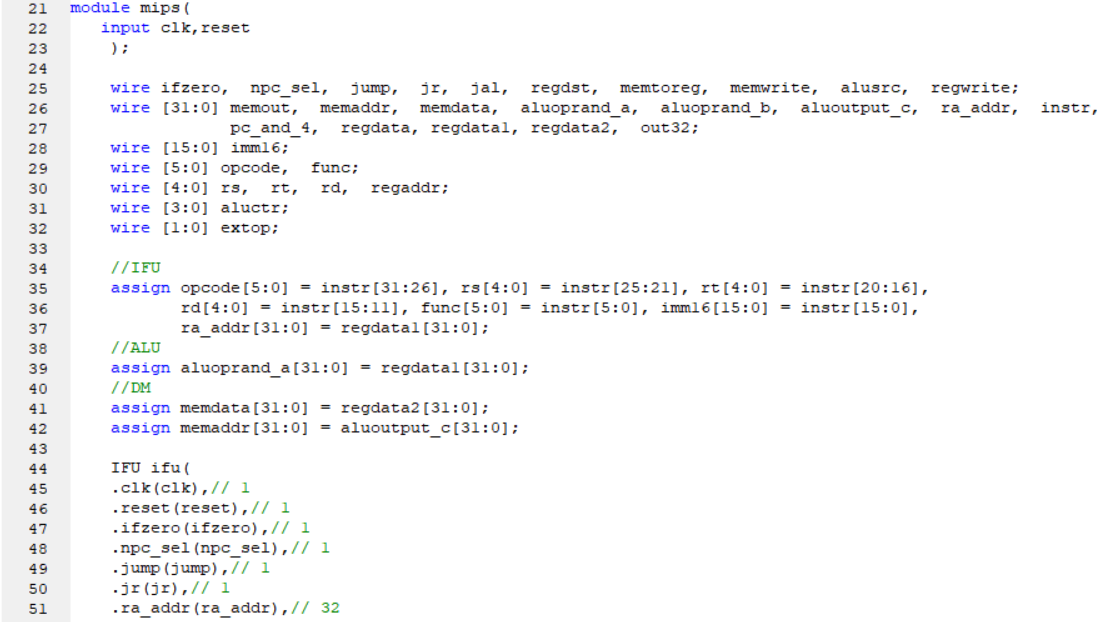
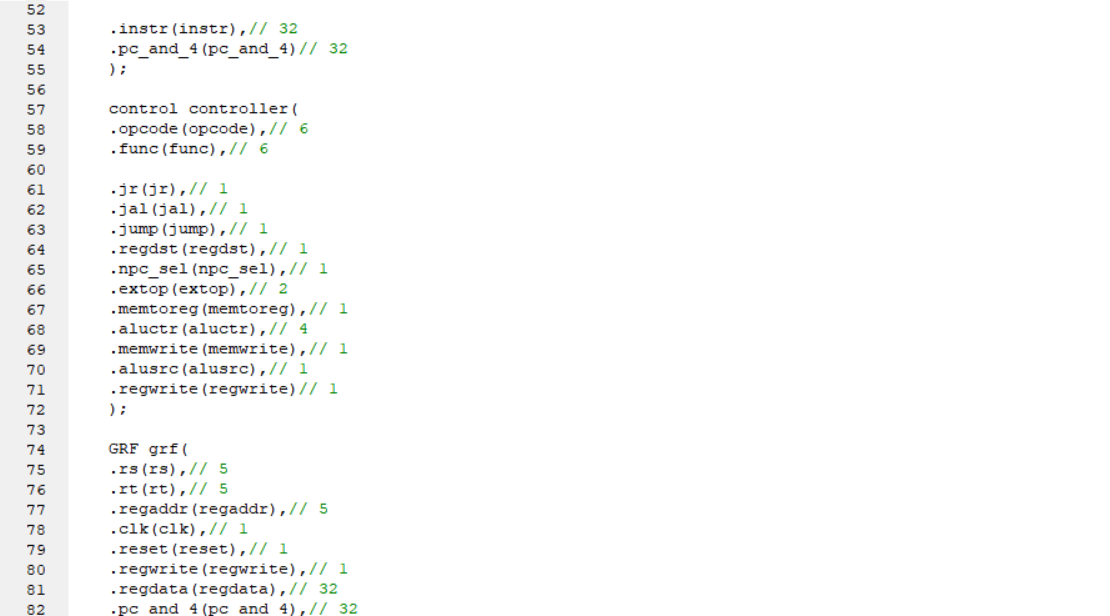


图1 Logisim顶层模块图





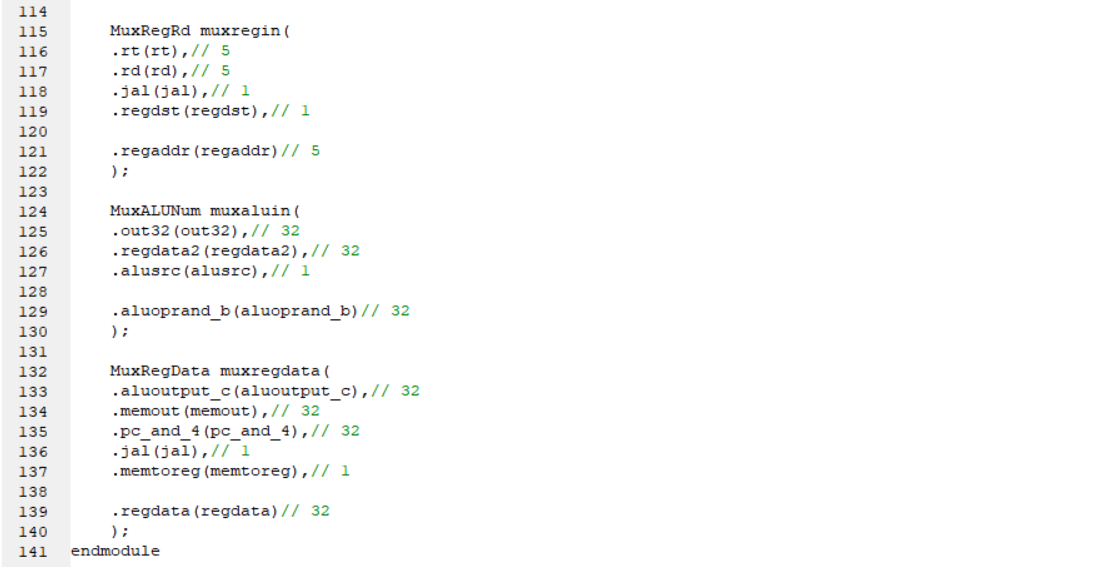
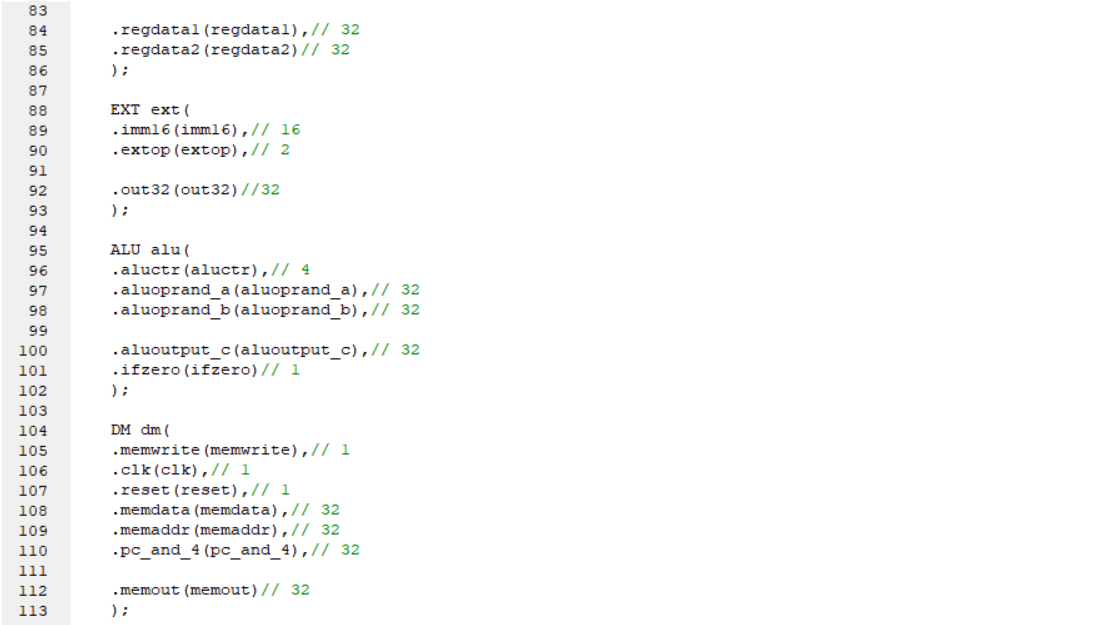


图2 顶层模块代码

1. **顶层多选器**

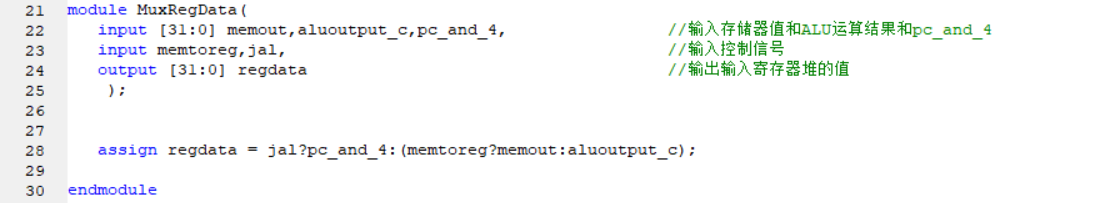
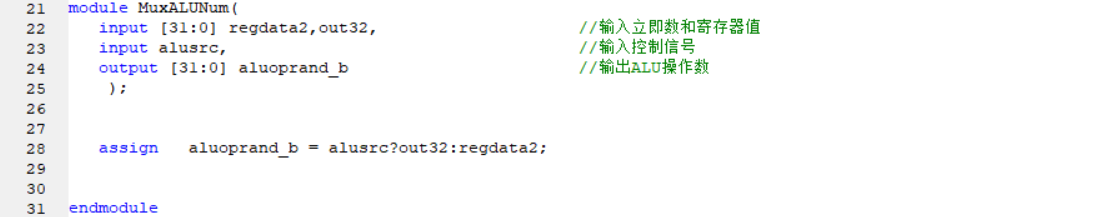
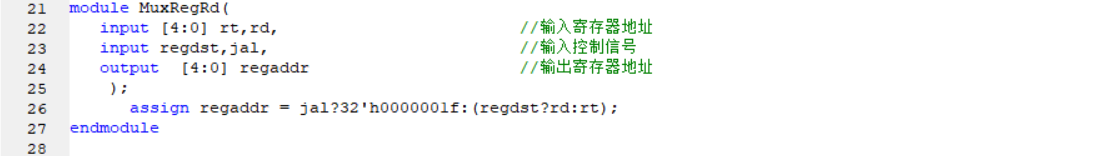


图3 顶层多选器代码

表1 MuxRegRd模块接口

|  |  |  |
| --- | --- | --- |
| **信号** | **方向** | **描述** |
| rt | I | 5位寄存器地址输入信号 |
| rd | I | 5位寄存器地址输入信号 |
| regdst | I | 寄存器地址选择信号 |
| jal | I | 寄存器地址选择信号，判断当前指令是否为jal |
| regaddr | O | 5位经选择后的寄存器地址输出信号 |

表2 MuxALUNum模块接口

|  |  |  |
| --- | --- | --- |
| **信号** | **方向** | **描述** |
| regdata2 | I | 32位读取的寄存器的运算数输入信号 |
| out32 | I | 32位经扩展后的运算数输入信号 |
| alusrc | I | 运算数选择信号 |
| aluoprand\_b | O | 32位经选择后的运算数输出信号，为ALU的操作数 |

表3 MuxRegData模块接口

|  |  |  |
| --- | --- | --- |
| **信号** | **方向** | **描述** |
| memout | I | 32位读取的DM的数据输入信号 |
| aluoutput\_c | I | 32位ALU运算结果的数据输入信号 |
| pc\_and\_4 | I | 32位当前指令地址加4后的输入信号 |
| memtoreg | I | 数据选择信号，判断是否从DM中读数据至GRF |
| jal | I | 数据选择信号，判断当前指令是否为jal |
| regdata | O | 32位经选择后的数据输出信号，为GRF的写入数据 |

表4 顶层多选器功能定义

|  |  |  |
| --- | --- | --- |
| **序号** | **多选器模块名称** | **功能描述** |
| 1 | MuxRegRd | jal有效则输出0x1f，否则  regdst有效则输出rd,否则  输出rt |
| 2 | MuxALUNum | alusrc有效则输出out32，否则  输出regdata2 |
| 3 | MuxRegData | jal有效则输出pc\_and\_4,否则  memtoreg有效则输出memout，否则  输出aluoutput\_c |

**二、模块定义**

1. **IFU**

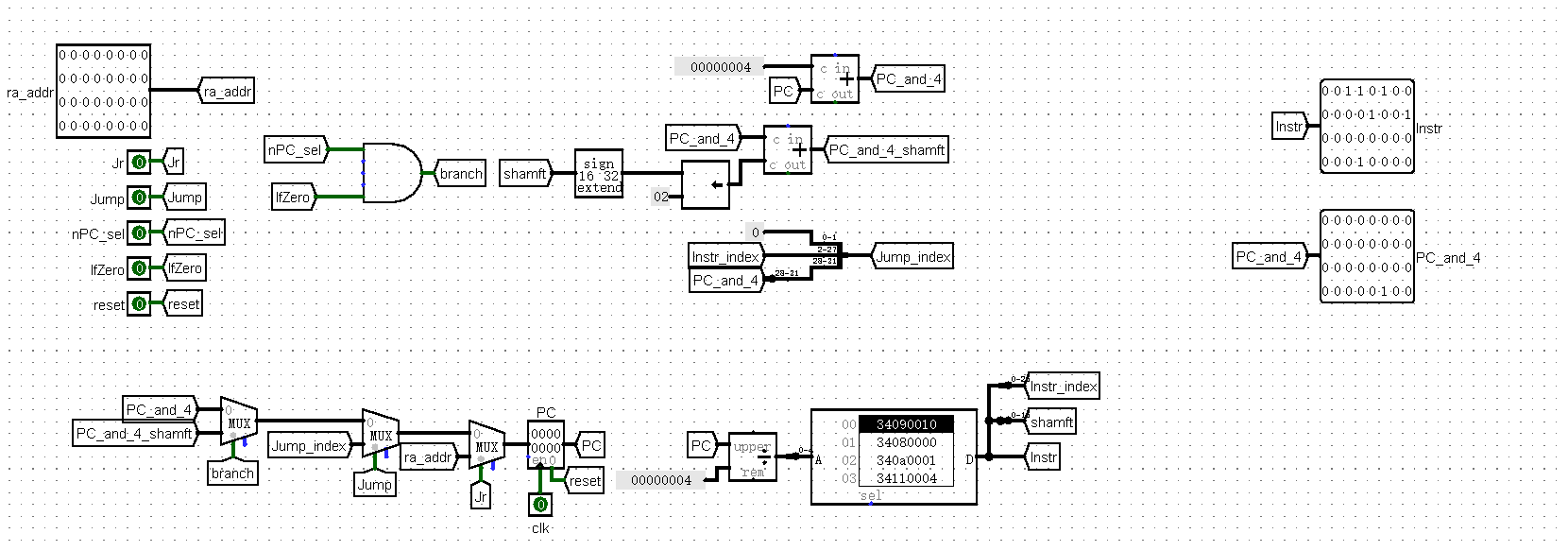
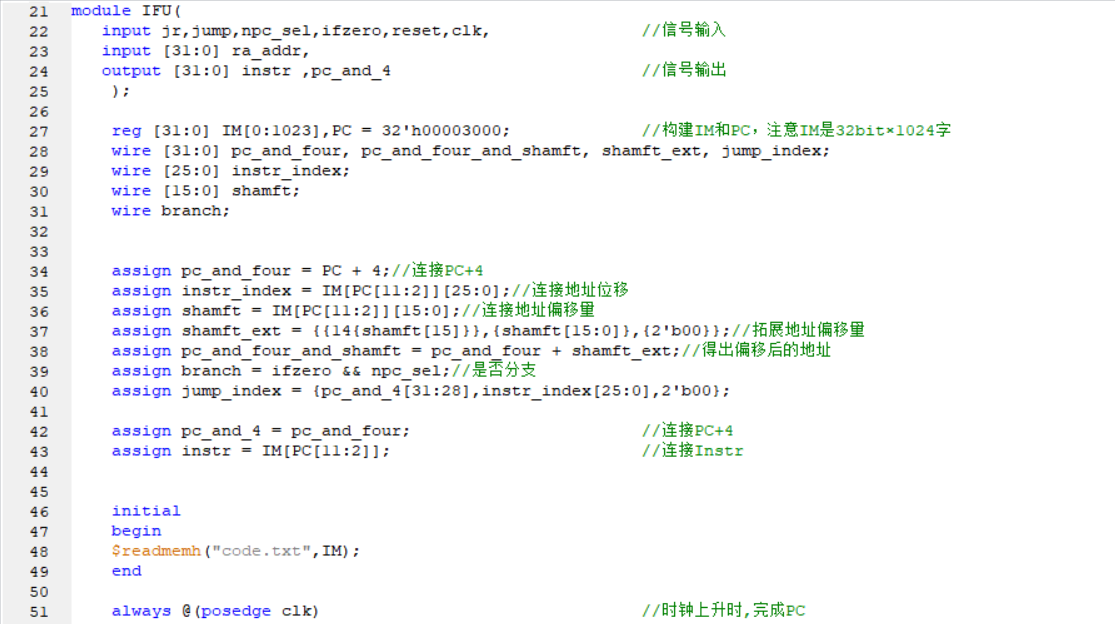


图4 Logisim IFU电路图



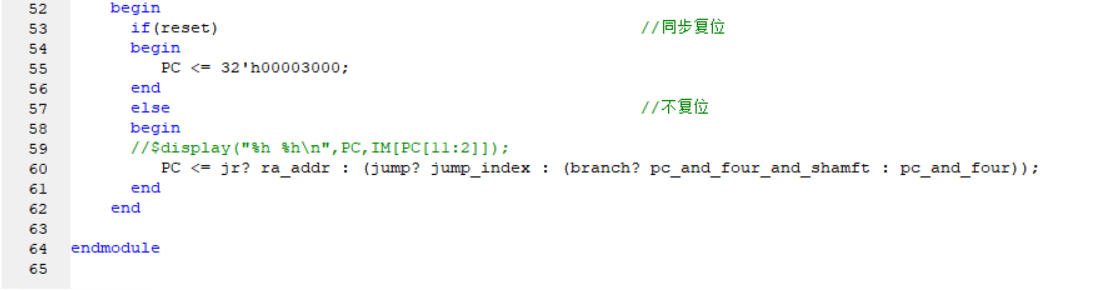


图5 IFU代码

**（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5 IFU模块接口

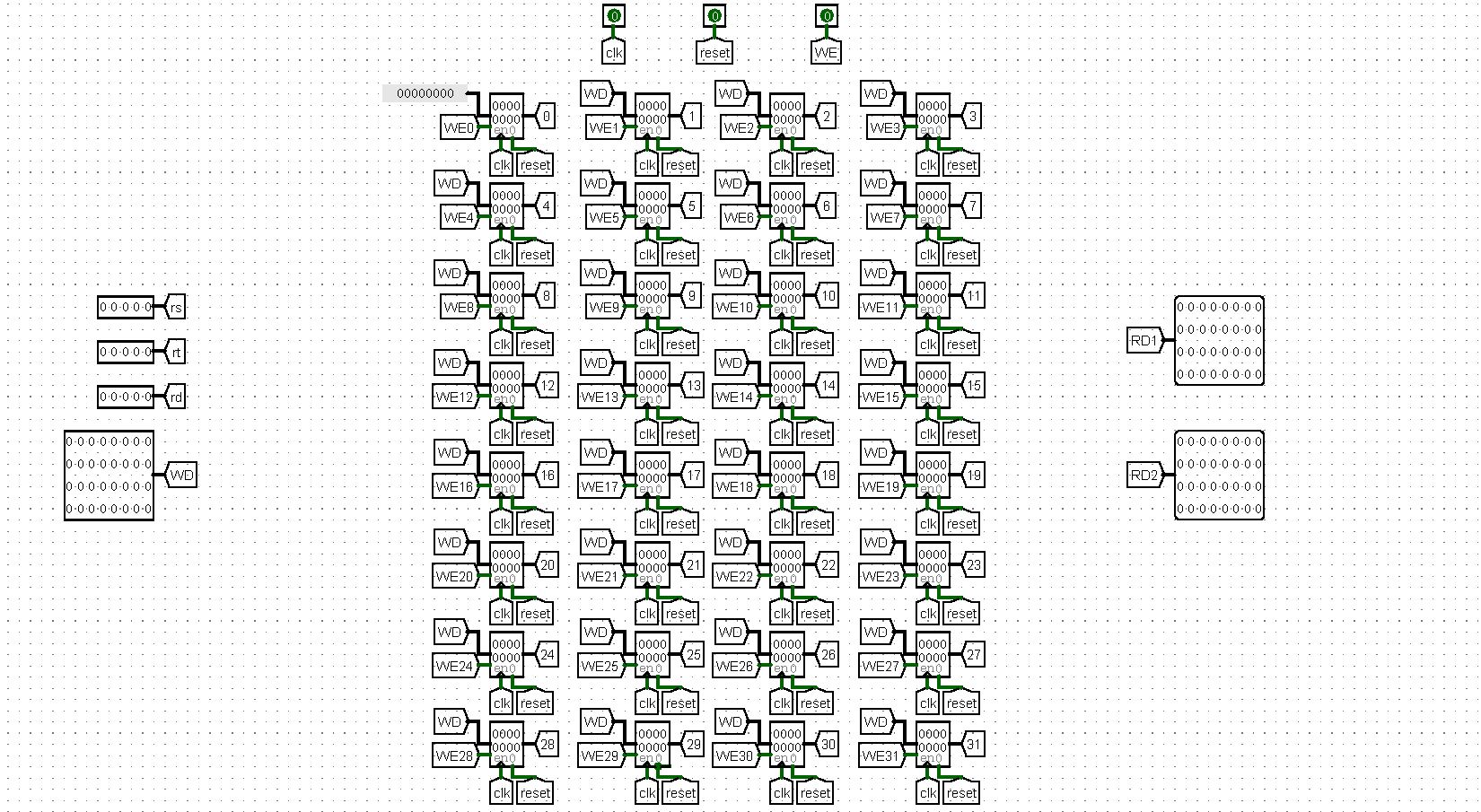
|  |  |  |
| --- | --- | --- |
| **信号名** | **方向** | **描述** |
| jr | I | 判断当前指令是否为jr指令的标志 |
| jump | I | 判断当前指令是否为跳转指令的标志 |
| npc\_sel | I | 判断当前指令是否为beq指令的标志  1：当前指令为beq指令  0：当前指令非beq指令 |
| ifzero | I | 判断ALU计算结果是否为0的标志  1：计算结果为0  0：计算结果非0 |
| ifbgez | I |  |
| ifbgtz | I |  |
| reset | I | 判断PC是否复位的信号  1：复位  0：无效 |
| clk | I | 时钟信号 |
| ra\_addr | I | 32位指令地址输入信号 |
| instr | O | 32位当前的MIPS指令 |
| pc\_and\_4 | O | 32位当前指令地址加4后的输出信号 |

1. **功能定义**

表6 IFU功能定义

|  |  |  |
| --- | --- | --- |
| **序号** | **功能名称** | **功能描述** |
| 1 | 复位 | 当reset信号有效时PC设置成0x0000\_3000 |
| 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  jr有效则PC←ra\_addr，否则  jump有效则PC←PC[31:28]||instr\_index||02，否则  ifzero和npc\_sel都有效则  PC←PC+4+sign\_ext(offset||02)否则  PC←PC+4 |

1. **GRF**



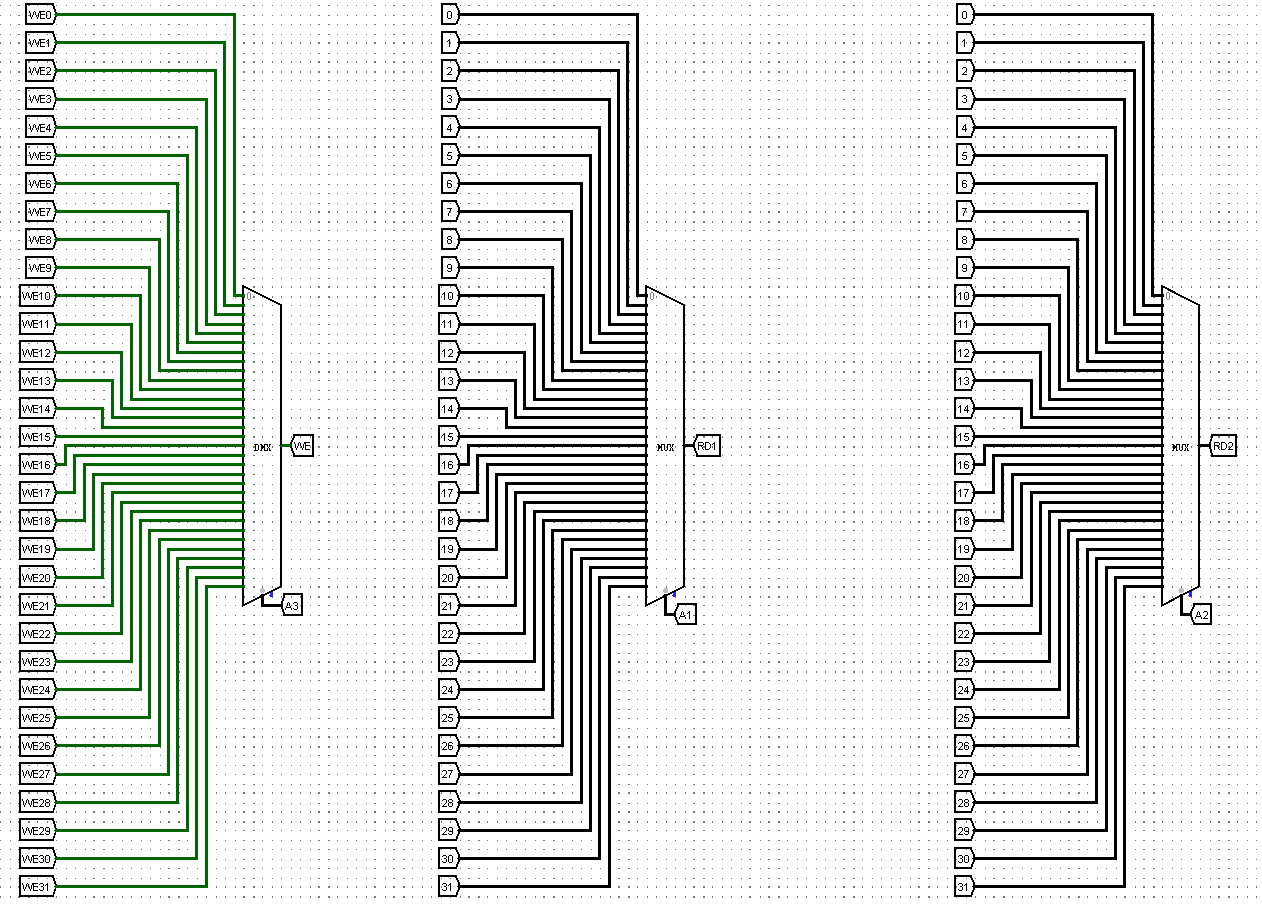


图6 Logisim GRF电路图

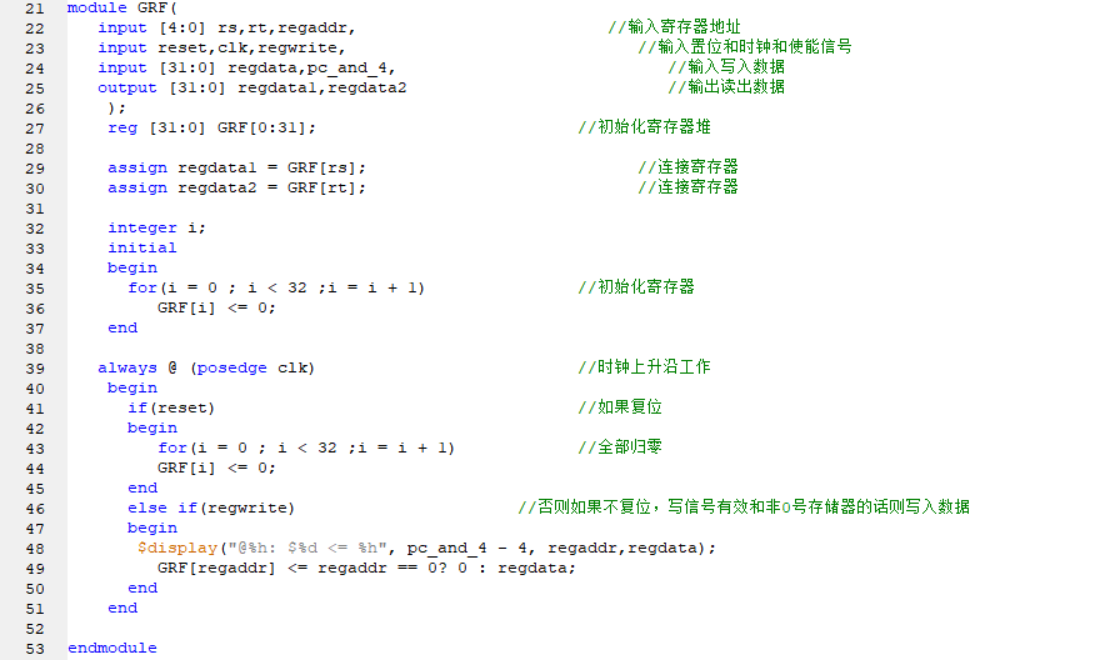


图7 GRF代码

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7 GRF模块接口

|  |  |  |
| --- | --- | --- |
| **信号名** | **方向** | **描述** |
| rs | I | 5位寄存器读地址输入信号 |
| rt | I | 5位寄存器读地址输入信号 |
| regaddr | I | 5位寄存器写地址输入信号 |
| reset | I | 复位信号  1：复位  0：无效 |
| clk | I | 时钟信号 |
| regwrite | I | 寄存器写使能信号 |
| regdata | I | 32位寄存器写数据 |
| pc\_and\_4 | I | 32位当前指令地址加4后的输入信号 |
| regdata1 | O | 32位rs地址对应的寄存器的数据的输出信号 |
| regdata2 | O | 32位rt地址对应的寄存器的数据的输出信号 |

1. **功能定义**

表8 GRF功能定义

|  |  |  |
| --- | --- | --- |
| **序号** | **功能名称** | **描述** |
| 1 | 读寄存器 | 读出rs、rt地址对应寄存器中所存储的数据到regdata1、regdata2 |
| 2 | 写寄存器 | 当regwrite信号有效且时钟上升沿到来时，将regdata写入regaddr所对应的寄存器中 |

1. **ALU**

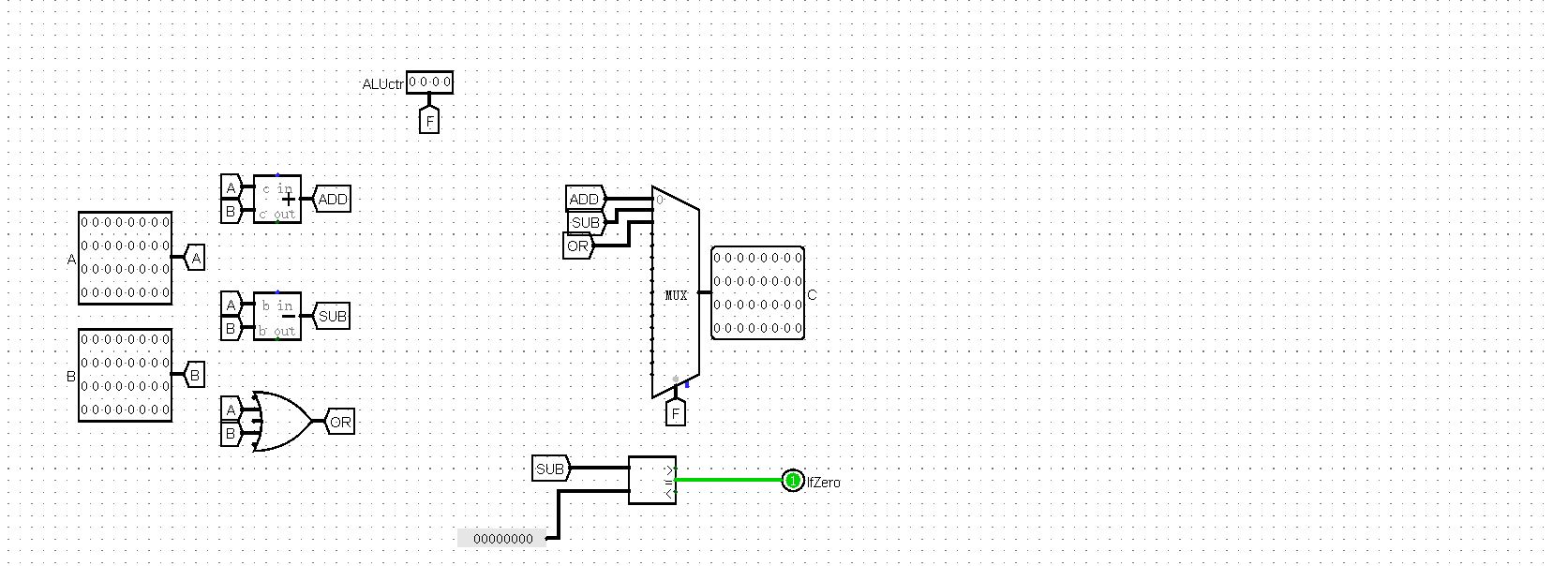


图8 Logisim ALU电路图

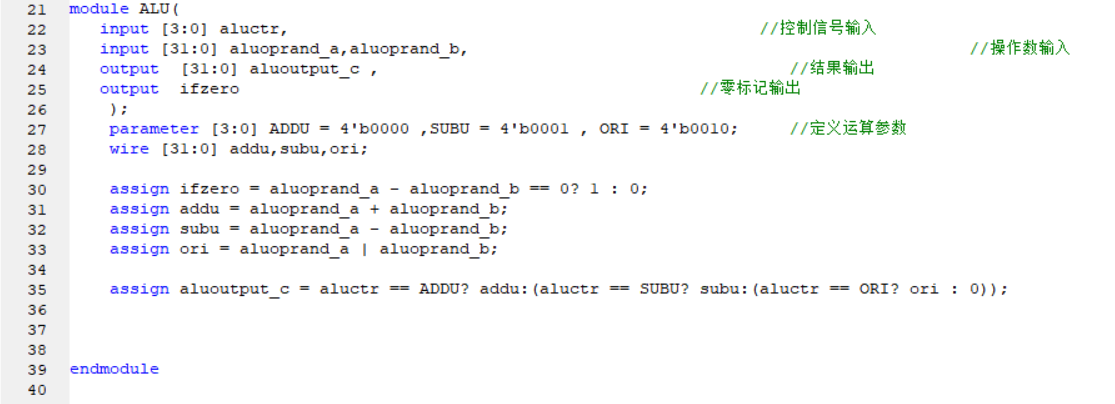


图9 ALU 代码

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9 ALU模块接口

|  |  |  |
| --- | --- | --- |
| **信号名** | **方向** | **描述** |
| aluctr | I | 4位输入信号，ALU的功能选择信号：  0000：ALU进行加法运算  0001：ALU进行减法运算  0010：ALU进行或运算 |
| aluoprand\_a | I | 32位输入信号，ALU的第一个操作数 |
| aluoprand\_b | I | 32位输入信号，ALU的第二个操作数 |
| aluoutput\_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 |
| ifbgez |  |  |
| ifbgtz |  |  |

1. **功能定义**

表10 ALU功能定义

|  |  |  |
| --- | --- | --- |
| **序号** | **功能名称** | **描述** |
| 1 | 加运算 | aluoutput\_c = aluoprand\_a + aluoprand\_b |
| 2 | 减运算 | aluoutput\_c = aluoprand\_a - aluoprand\_b |
| 3 | 或运算 | aluoutput\_c = aluoprand\_a | aluoprand\_b |

1. **EXT**

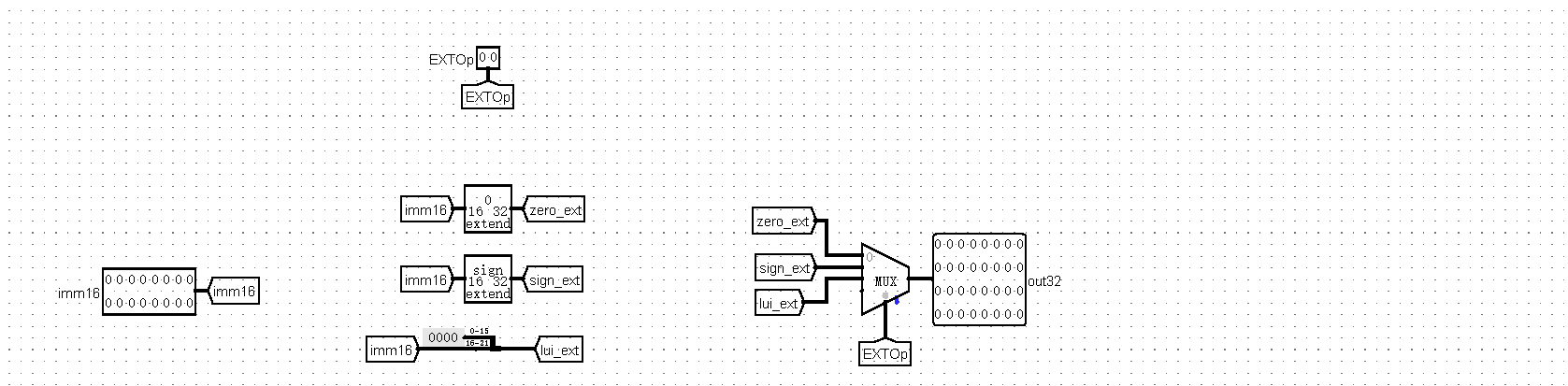


图10 Logisim EXT电路图

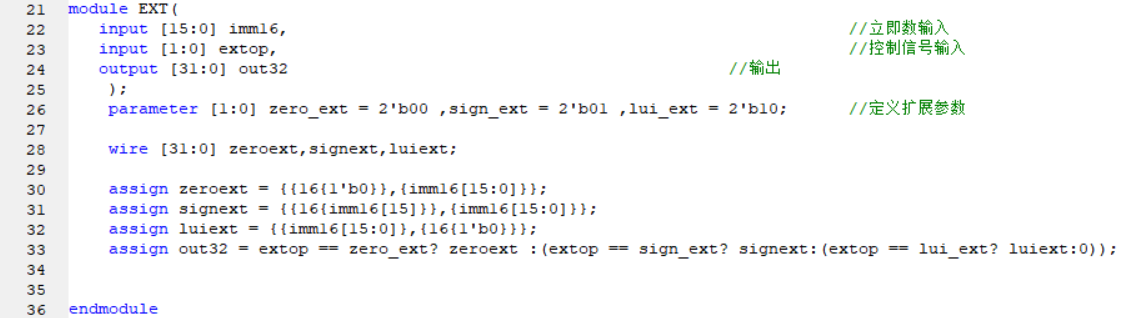


图11 EXT 代码

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11 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12 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**

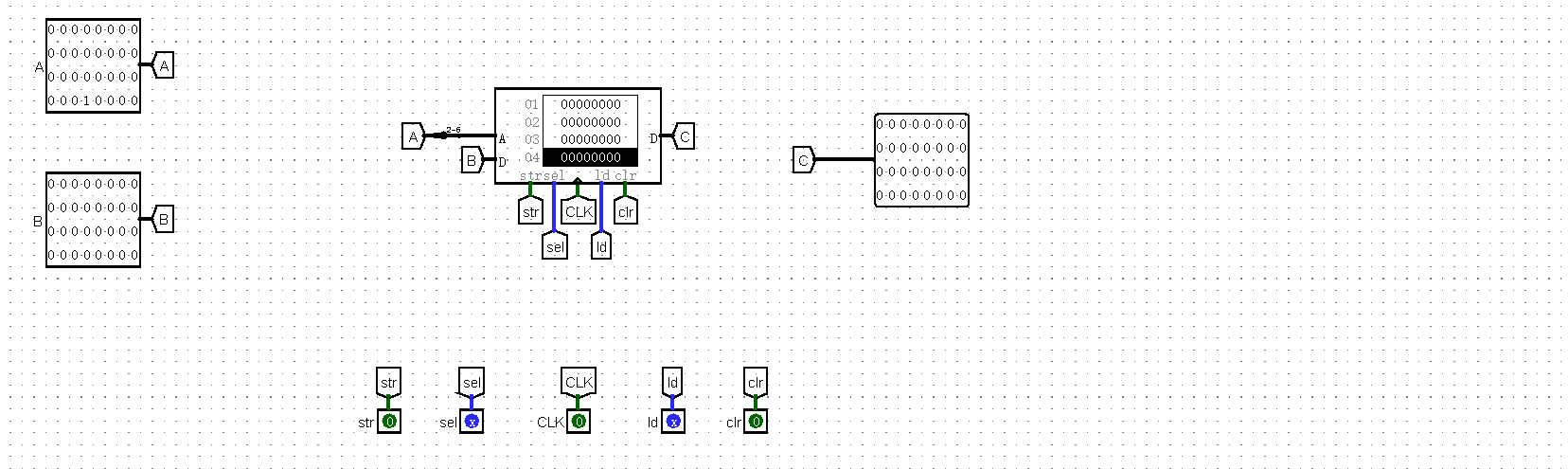


图12 Logisim DM电路图



图13 DM代码

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13 DM模块接口

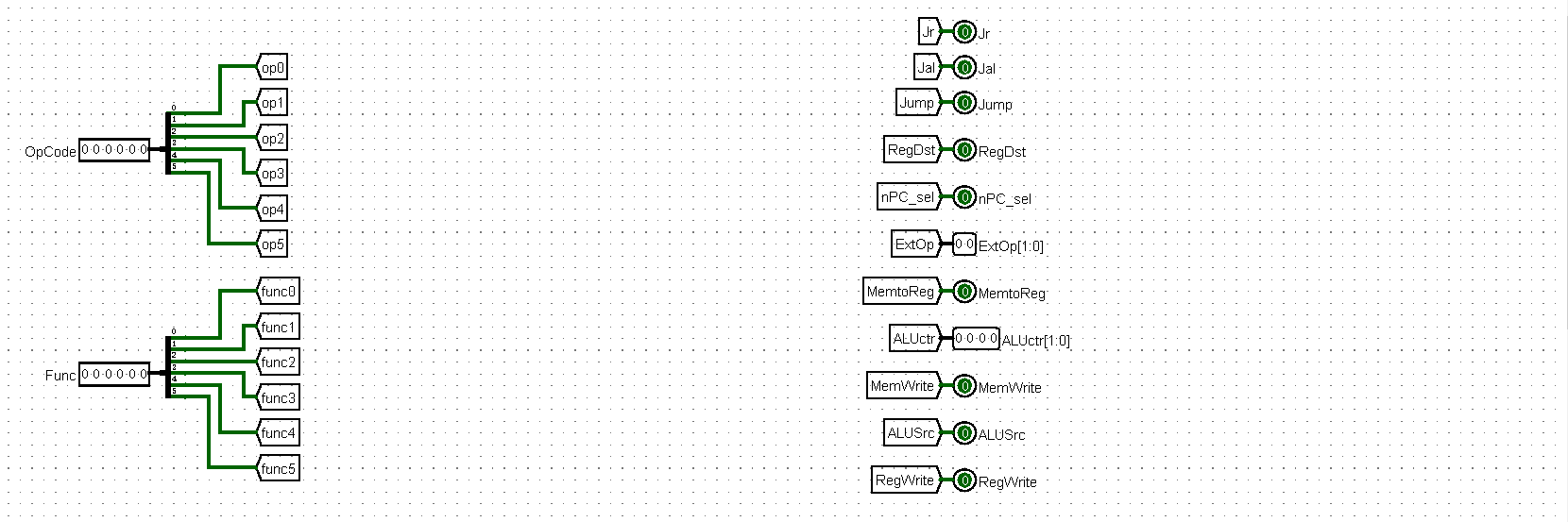
|  |  |  |
| --- | --- | --- |
| **信号名** | **方向** | **描述** |
| memaddr | I | 32位输入信号，操作存储器的地址 |
| memdata | I | 32位输入信号，为写入数据的输入 |
| pc\_and\_4 | I | 32位当前指令地址加4后的输入信号 |
| memwrite | I | 读写控制信号  1：写信号  0：读信号 |
| clk | I | 时钟信号 |
| reset | I | 复位信号 |
| memout | O | 32位输出信号，输出存储器操作地址输入对应的数据 |

1. **功能定义**

表14 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 |
| 3 | 复位 | 当reset有效时将存储器所有数据置0 |

**三、Controller设计**



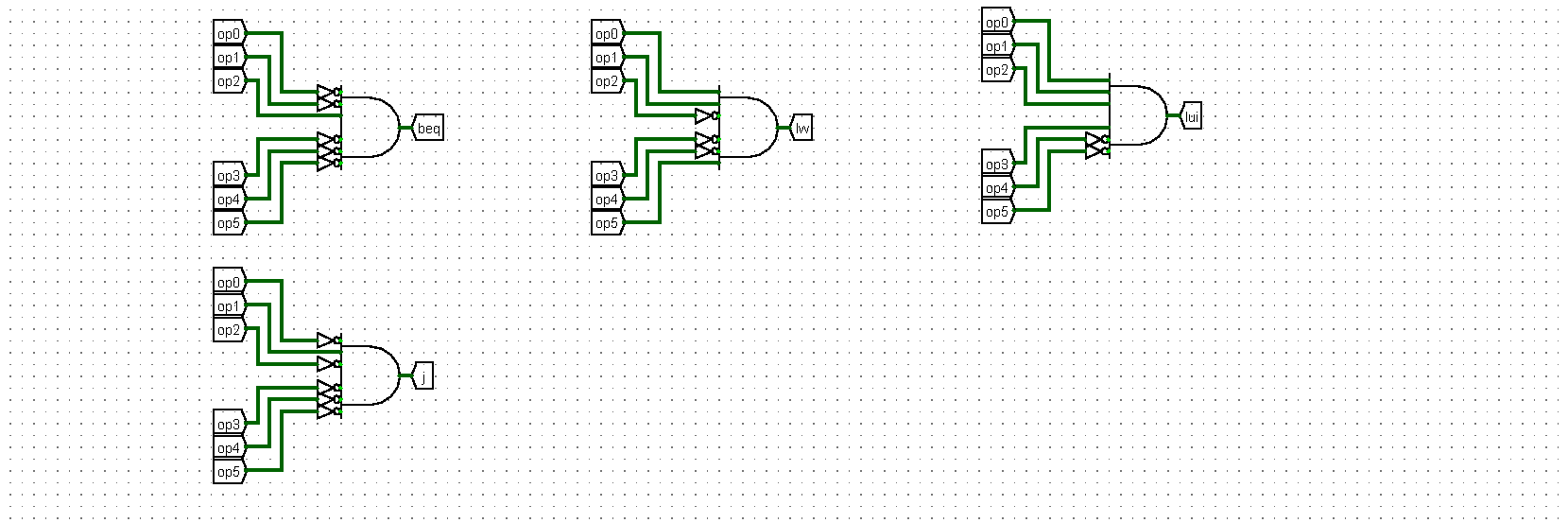
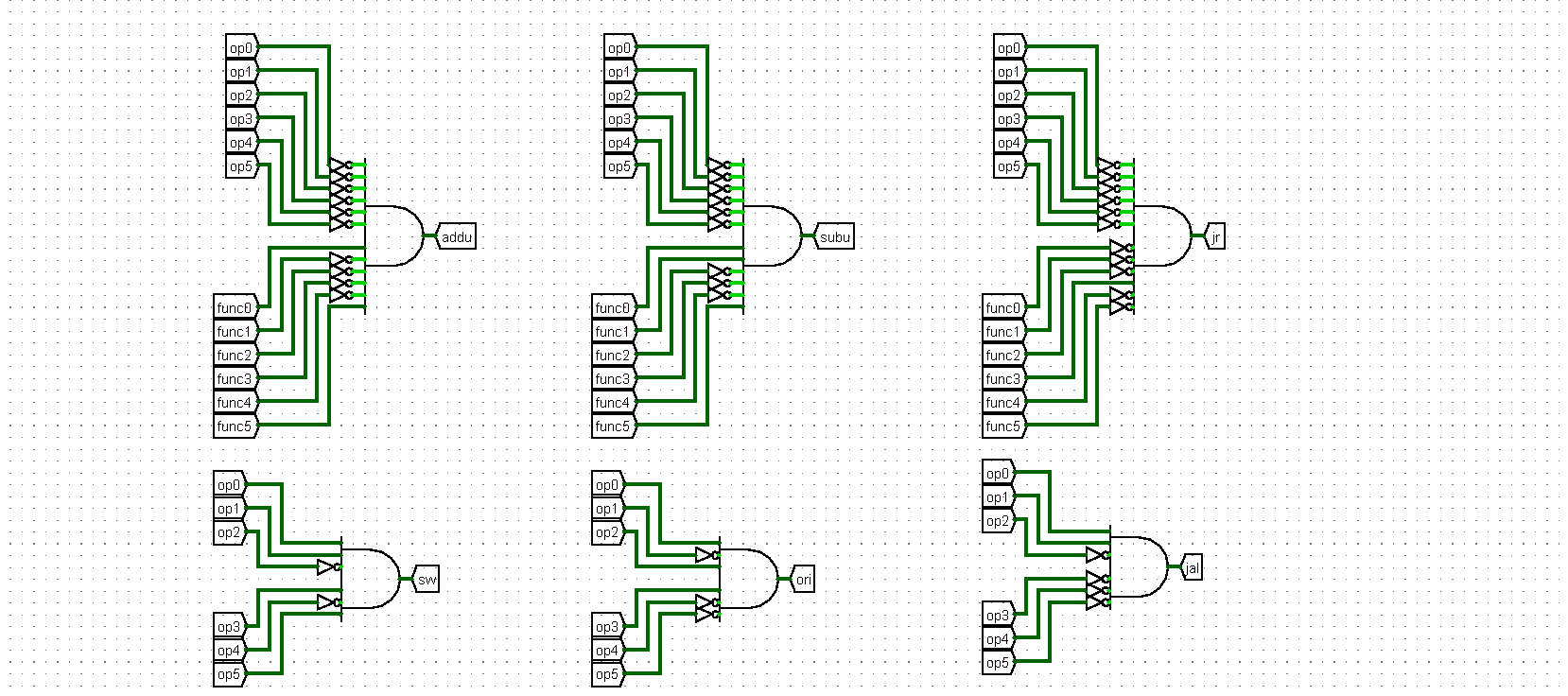
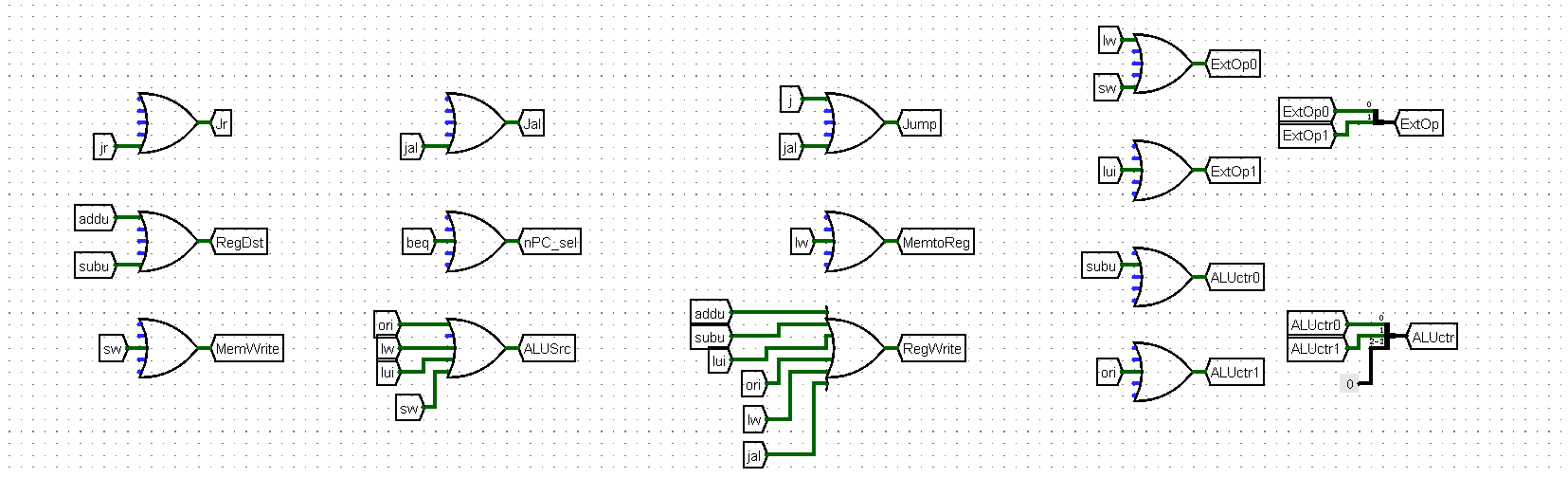


图14 Logisim Controller电路图

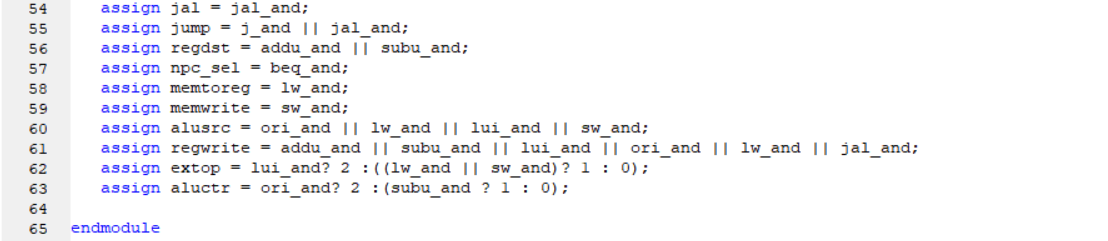
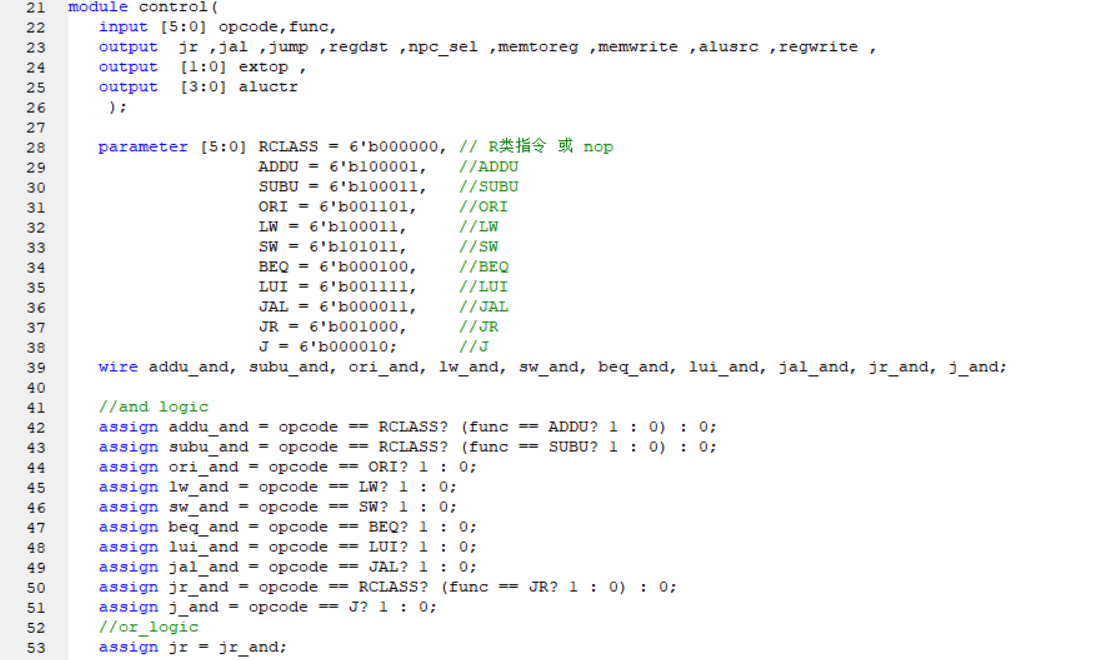


图15 Controller 代码

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15 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 |
| jr | O | JR指令标志 |
| jal | O | JAL指令标志 |
| jump | O | JUMP指令标志 |
| regdst | O | GRF写地址控制 |
| npc\_sel | O | BEQ指令标志 |
| memtoreg | O | DM输出数据控制标志 |
| 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 |
| extop | O | EXT扩展方式控制标志 |
| aluctr | O | ALU运算控制标志 |

1. **单周期真值表**

表16 Controller单周期真值表

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **op**  **code** | **func** | **jr** | **jal** | **jump** | **reg**  **dst** | **npc**  **\_sel** | **mem**  **toreg** | **memwrite** | **alusrc** | **reg**  **write** | **extop** | **aluctr** |
| **addu** | **000000** | **100001** | **0** | **0** | **0** | **1** | **0** | **0** | **0** | **0** | **1** | **X** | **0** |
| **subu** | **000000** | **100011** | **0** | **0** | **0** | **1** | **0** | **0** | **0** | **0** | **1** | **X** | **1** |
| **ori** | **001101** | **N/A** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **1** | **1** | **0** | **2** |
| **lw** | **100011** | **N/A** | **0** | **0** | **0** | **0** | **0** | **1** | **0** | **1** | **1** | **1** | **0** |
| **sw** | **101011** | **N/A** | **0** | **0** | **0** | **X** | **0** | **0** | **1** | **1** | **0** | **1** | **0** |
| **beq** | **000100** | **N/A** | **0** | **0** | **0** | **X** | **1** | **0** | **0** | **0** | **0** | **X** | **0** |
| **j** | **000010** | **N/A** | **0** | **0** | **1** | **X** | **0** | **0** | **0** | **X** | **0** | **X** | **0** |
| **jr** | **000000** | **001000** | **1** | **0** | **0** | **X** | **0** | **0** | **0** | **X** | **0** | **X** | **0** |
| **jal** | **000011** | **N/A** | **0** | **1** | **1** | **0** | **0** | **0** | **0** | **X** | **1** | **X** | **0** |
| **lui** | **001111** | **N/A** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **1** | **1** | **2** | **0** |
| **bgez** | **000001** | **00001**  **rt** | **0** | **0** | **0** | **X** | **1** | **0** | **0** | **0** | **0** | **X** | **0** |
| **bgtz** | **000111** | **N/A** | **0** | **0** | **0** | **X** | **1** | **0** | **0** | **0** | **0** | **X** | **0** |
| **blez** | **000110** | **N/A** |  |  |  |  |  |  |  |  |  |  |  |
| **bltz** | **000001** | **00000**  **rt** |  |  |  |  |  |  |  |  |  |  |  |
|  | **op**  **code** | **func** | **jr** | **jal** | **jump** | **reg**  **dst** | **npc**  **\_sel** | **mem**  **toreg** | **memwrite** | **alusrc** | **reg**  **write** | **extop** | **aluctr** |
| **bne** | **000101** | **N/A** |  |  |  |  |  |  |  |  |  |  |  |
| **lw** | **100011** | **N/A** | **0** | **0** | **0** | **0** | **0** | **1** | **0** | **1** | **1** | **1** | **0** |
| **lb** |  | **N/A** | **0** | **0** | **0** | **0** | **0** | **1** | **0** | **1** | **1** | **1** | **0** |
| **lbu** |  | **N/A** | **0** | **0** | **0** | **0** | **0** | **1** | **0** | **1** | **1** | **1** | **0** |
| **lh** |  | **N/A** | **0** | **0** | **0** | **0** | **0** | **1** | **0** | **1** | **1** | **1** | **0** |
| **lhu** |  | **N/A** | **0** | **0** | **0** | **0** | **0** | **1** | **0** | **1** | **1** | **1** | **0** |
| **sb** |  | **N/A** | **0** | **0** | **0** | **X** | **0** | **0** | **1** | **1** | **0** | **1** | **0** |
| **sh** |  | **N/A** | **0** | **0** | **0** | **X** | **0** | **0** | **1** | **1** | **0** | **1** | **0** |
| **sll** |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **sllv** |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **srl** |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **srlv** |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |

1. **测试CPU**

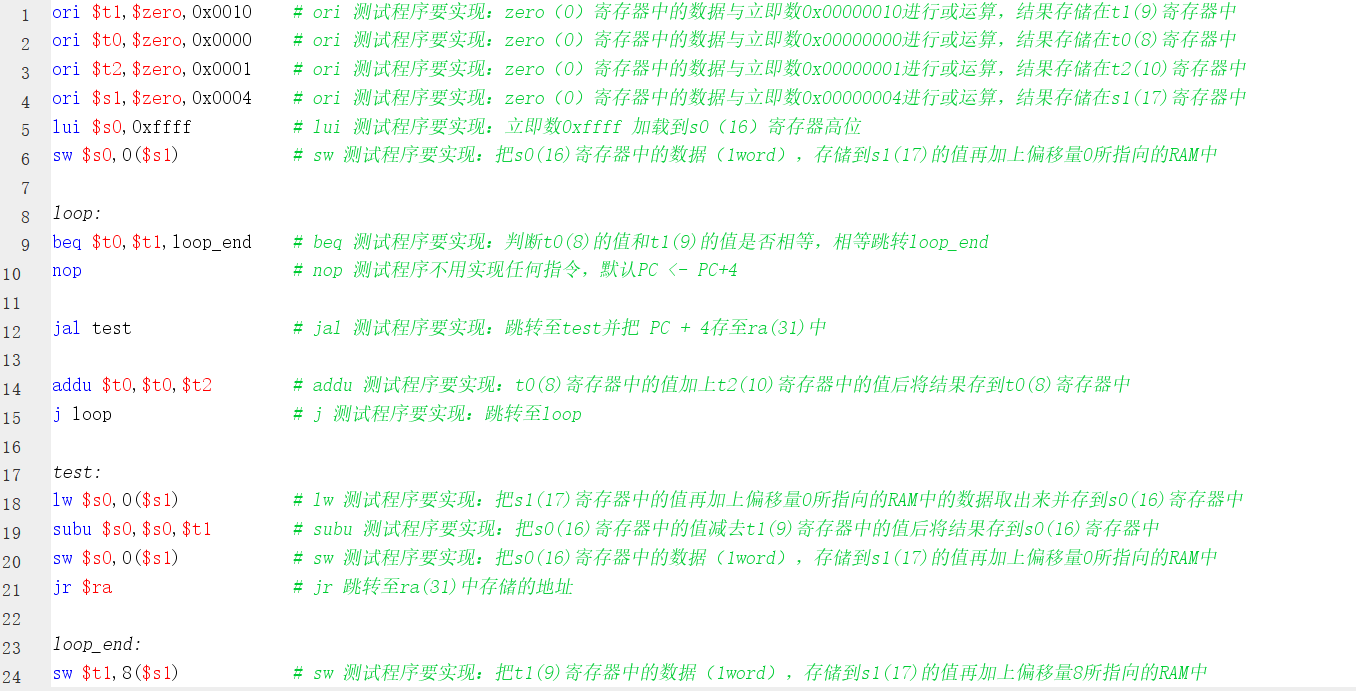


图16 code 测试代码



图17 code 测试机械码

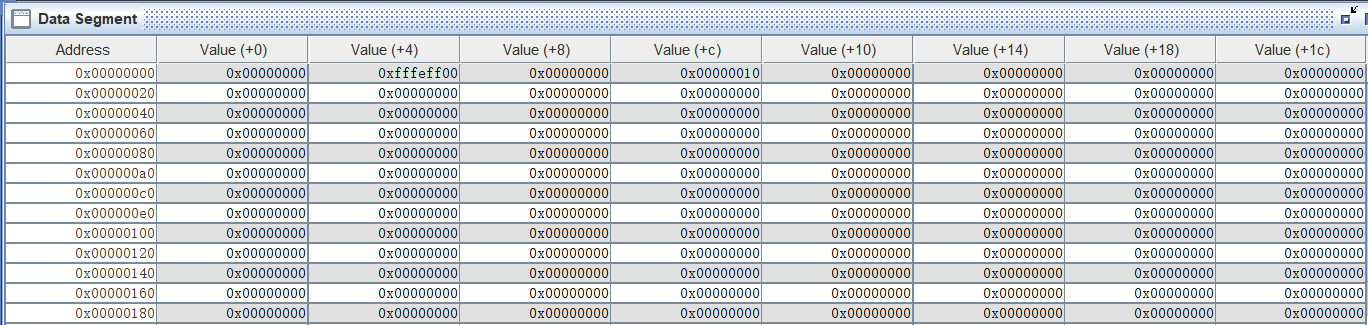


图18 MARS 模拟结果之DM

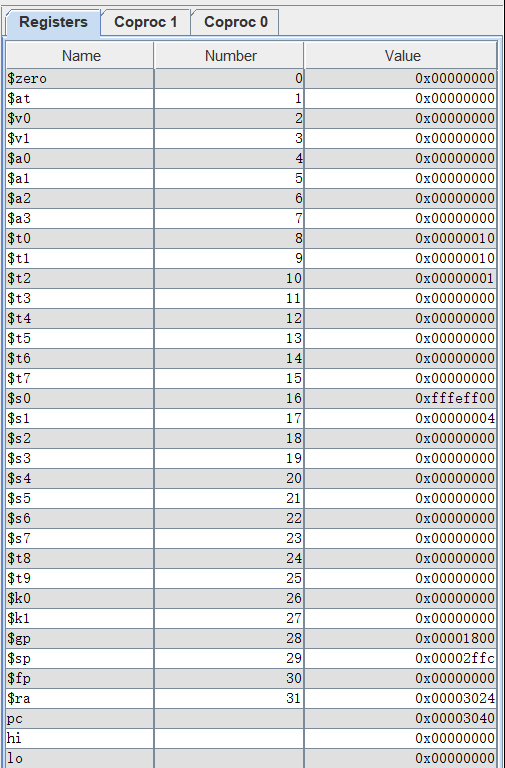


图19 MARS 模拟结果之GRF

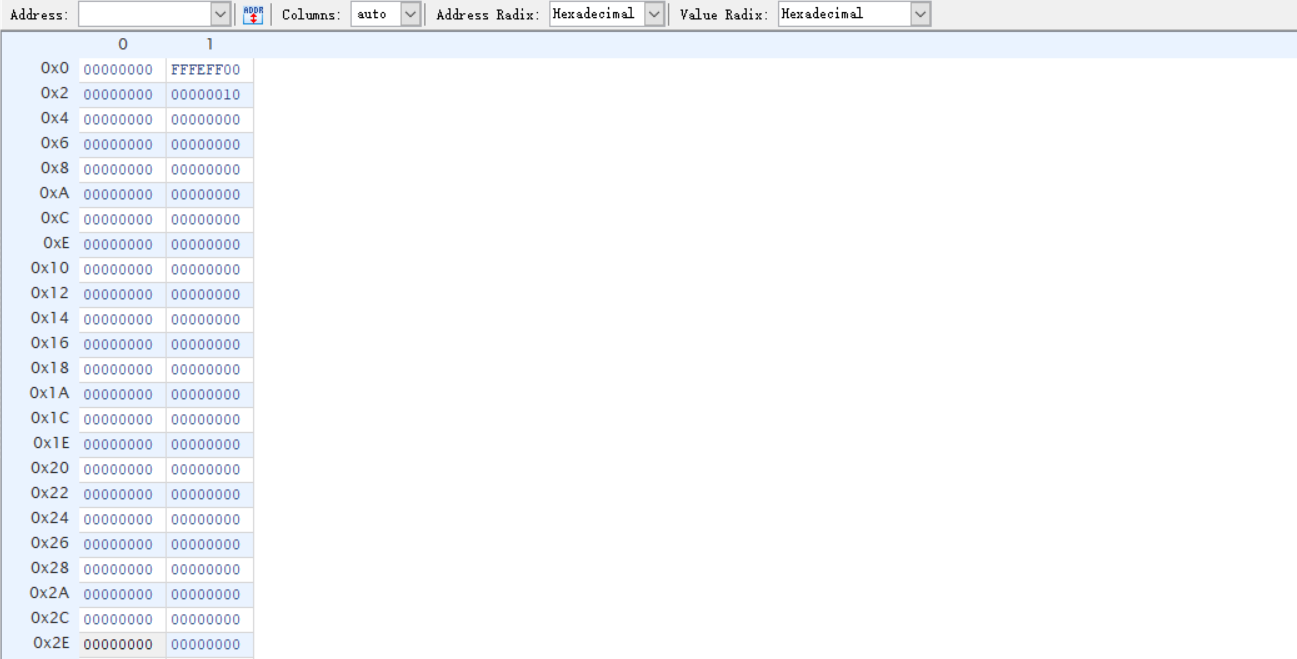


图20 ISIM 仿真结果之DM

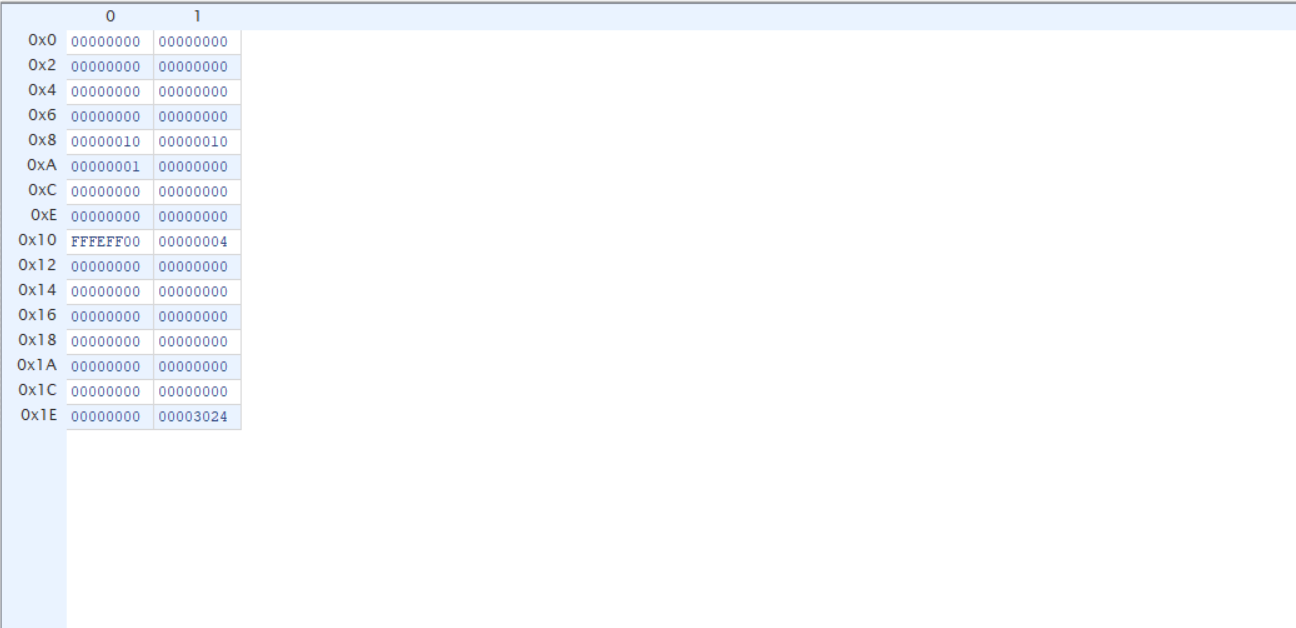


图21 ISIM 仿真结果之GRF

jal test:

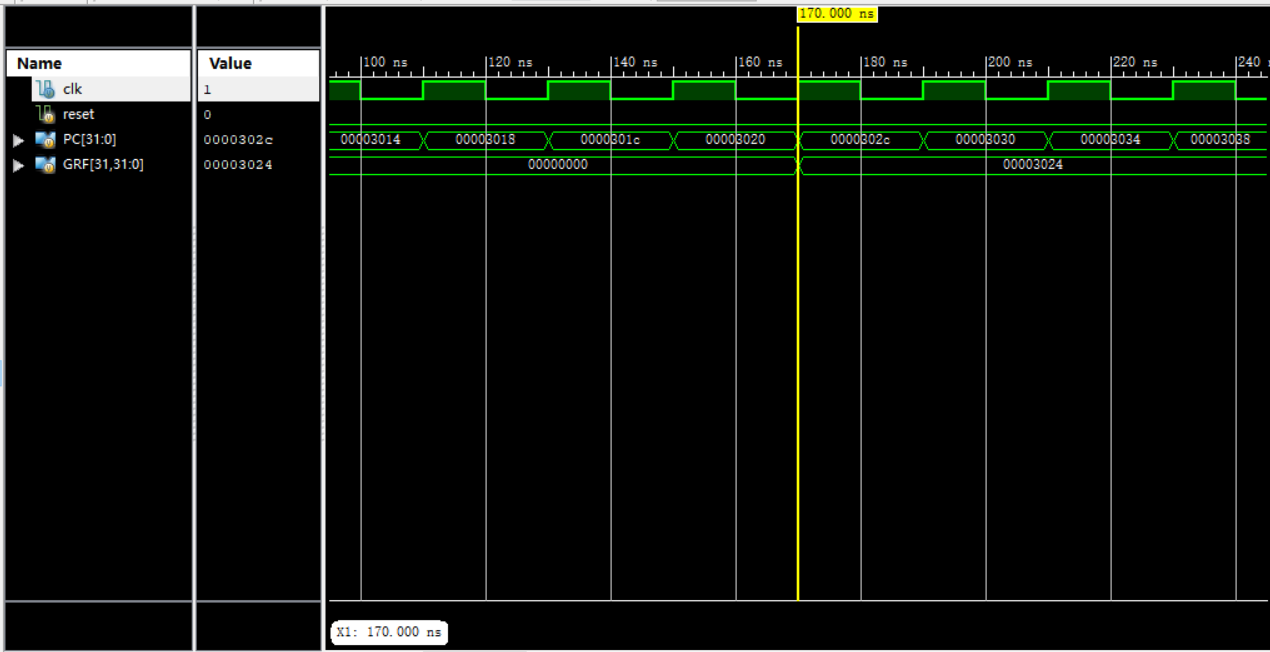
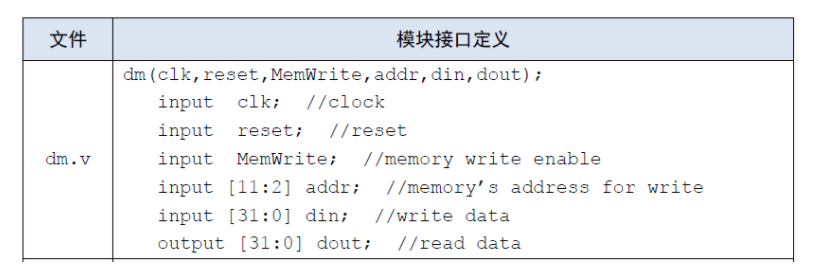


图22 ISIM 仿真结果之波形

1. **思考题**
2. **根据你的理解，在下面给出的DM的输入示例中，地址信号addr位数为什么是[11:2]而不是[9:0]？这个addr信号又是从哪里来的？**

MIPS中内存地址按字节编址，地址信号为4的倍数，而设置的CPU内存按字编址，所以取[11:2]相当于除以4，即把字节的地址转成字的地址。addr信号从ALU模块中来。

1. **在相应的部件中，**reset的优先级**比其他控制信号（不包括clk信号）都要**高**，且相应的设计都是**同步复位**。清零信号reset是针对哪些部件进行清零复位操作？这些部件为什么需要清零？**

reset是对GRF和DM进行清零复位操作。因为首指令的地址是0x000030000，所以PC不能进行清零复位操作。而GRF和DM是存储数据的单元，复位时应全部清零。

1. **列举出用Verilog语言设计控制器的几种编码方式（至少三种），并给出代码示例。**

宏定义define：

`define lw = opcode == 100011? 1 : 0;

三目运算符+assign：

assign lw = opcode == 100011? 1 : 0;

case选择：

always @ (\*)begin

case(opcode)

6’b000000:

case(func)

6’b100011:

addu <= 1;

endcase

6’b100011:

lw <= 1;

endcase

end

1. **根据你所列举的编码方式，说明他们的优缺点。**

宏定义define:

优点：组合逻辑描述，清晰直观，便于理解

缺点：多位信号如extop的判断繁琐

case选择：

优点：清晰直观，便于添加指令

缺点：写在always块中，与controller纯组合逻辑描述有出入，可能会产生未知错误

三木运算符+assign:

优点：能很好地描述logisim中的controller的与或逻辑部分，描述的结构一致

缺点：多位信号如extop的判断繁琐

1. **C语言是一种弱类型程序设计语言。C语言中不对计算结果溢出进行处理，这意味着C语言要求程序员必须很清楚计算结果是否会导致溢出。因此，如果仅仅支持C语言，MIPS指令的所有计算指令均可以忽略溢出。 请说明为什么在忽略溢出的前提下，addi与addiu是等价的，add与addu是等价的。提示：阅读《MIPS32® Architecture For Programmers Volume II: The MIPS32® Instruction Set》中相关指令的Operation部分 。**

查阅资料得到addi和addiu、add和addu的operation部分如下：

addi:

temp ← (GPR[rs]31||GPR[rs]31..0) + sign\_extend(immediate)

if temp32 ≠ temp31 then

SignalException(IntegerOverflow)

else

GPR[rt] ← temp

endif

addiu:

temp ← GPR[rs] + sign\_extend(immediate)

GPR[rt]← temp

add:

temp ← (GPR[rs]31||GPR[rs]31..0) + (GPR[rt]31||GPR[rt]31..0)

if temp32 ≠ temp31 then

SignalException(IntegerOverflow)

else

GPR[rd] ← temp

endif

addu:

temp ← GPR[rs] + GPR[rt]

GPR[rd] ← temp

可以看出在忽略溢出的情况下，addi和addiu执行的操作都是

temp ← GPR[rs] + sign\_extend(immediate)

GPR[rt]← temp

add和addu执行的操作都是

temp ← GPR[rs] + GPR[rt]

GPR[rd] ← temp

所以在忽略溢出的情况下，addi和addiu、add和addu是等价的

1. **根据自己的设计说明单周期处理器的优缺点。**

优点：每个单周期执行一条指令，不会像多周期一样产生冒险，设计简单

缺点：因为每个单周期执行一条指令，故处理器的周期由处理器支持的指令集中的指令的最长路径决定，会导致程序执行效率低

**7、简要说明jal、jr和堆栈的关系。**

jal与jr成对出现，每次执行jal指令，都会把下一条指令的地址PC+4存进ra中，相当于压栈，然后跳转；而每次执行jr指令，程序则跳转回ra存储的地址对应的指令，相当于出栈。意即当调用多次jal与jr时，我们总会把之前的ra值存在内存中，然后jal存进ra对应的地址总是最后的地址，即存进栈顶元素，而使用jr时，则跳转至当前的ra对应的地址，即取出栈顶元素，符合栈的LIFO原则。