



**计算机组成原理课 程 设 计**

团队实验报告

|  |  |
| --- | --- |
| 学 院 | 计算机学院 |
| 专 业 | 计算机科学与技术 |
| 指导老师 |  |
| 组 长 | 石宏岩 |
| 组 员 | 夏梓茗、陈雪峰、解飞 |
| 组长联系方式 |  |

二O二三年 9月

**目 录**

1.项目简述

2.组员分工

3.设计目的

4.设计环境

5.设计原理及内容

6.设计与实现

7.测试

8.问题及解决方法

9.心得体会及总结

# 项目简述

实现经典五级流水线CPU，支持34条riscv指令，包含冒险冲突处理，结合汇编与接口设计实验，合成一个能通过cpu通过vga接口显示数字的系统。

# 组员分工（团队报告，个人报告不需要）

小组分工见表2.1 小组分工。

|  |  |
| --- | --- |
| **组员** | **分工任务** |
| 组长 石宏岩 | * 上板的完成及测试、vga程序的撰写和测试以及综合上板测试 |
| 组员1 夏梓茗 | * 流水线cpu的设计和实现、帮助完善数据冒险和控制冒险、协助测试和完善代码 |
| 组员2 解飞 | * 数据冒险和控制冒险的处理以及分支预测 |
| 组员3 陈雪峰 | * 数据冒险和控制冒险的处理以及分支预测 |

表2.1 小组分工

# 设计目的

基于riscv指令集在上学期单周期cpu的基础上进行流水线cpu的设计，采用基础的五级流水线思路，提升cpu的性能效率，增加对cpu架构的理解，活用计算机组成原理的知识。

# 设计环境

|  |  |
| --- | --- |
| 操作系统 | Windows10 专业版 |
| 编程语言 | Verilog |
| EDA工具 | Vivado2019.2 |
| 汇编语言 | Riscv |
| 汇编程序编辑器 | Vscode2022 |

# 设计原理及内容

## CPU整体架构

本流水线CPU覆盖34条指令(见表5.1-1)，下面将从数据通路和控制逻辑阐述设计思路。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 指令 | opcode | 功能 | |
| 1 | ADD | 0110011 | rd=rs+rt | |
| 2 | ADDI | 0110011 | rd=rs+imme | |
| 3 | SUB | 0010011 | rd=rs-rt | |
| 4 | SUBI | 0010011 | rd=rs- imme | |
| 5 | SLT | 0110011 | rd=(rs<rt)?1:0 | |
| 6 | SLTI | 0010011 | rd=(rs<im)?1:0 | |
| 7 | SLTU | 0110011 | rd=(rs<rt)?1:0 (无符号数) | |
| 8 | SLTIU | 0010011 | rd=(rs<im)?1:0（无符号数） | |
| 9 | AND | 0110011 | rd=rs&rt | |
| 10 | ANDI | 0010011 | rd=rs&im | |
| 11 | OR | 0110011 | rd=rs|rt | |
| 12 | ORI | 0010011 | rd=rs|im | |
| 13 | XOR | 0110011 | rd=rs xor rd | |
| 14 | XORI | 0010011 | rd=rs xor im | |
| 15 | SLL | 0110011 | rd=rt<<rs | |
| 16 | SLLI | 0010011 | rd=rt<<imme | |
| 17 | SRL | 0110011 | rd=rt>>rs | |
| 18 | SRLI | 0010011 | rd=rt>>imme | |
| 19 | SRA | 0110011 | rd=rt>>rs（符号位保留） | |
| 20 | SRAI | 0010011 | rd=rt>>imme（符号位保留） | |
| 21 | LUI | 0110111 | rt=im\*65536 | |
| 22 | BEQ | 1100011 | PC=(rs==rt)?PC+im<<2:PC | |
| 23 | BNE | 1100011 | PC=(rs!=rt)?PC+im<<2:PC | |
| 24 | BGE | 1100011 | PC=(rs>=rt)?PC+im<<2:PC | |
| 25 | BGEU | 1100011 | PC=(rs>=0)?PC+im<<2:PC | |
| 26 | BLTU | pc = (rs<0)pc+offset<<2:pc | |
| 27 | JAL | $31=PC;PC={(PC+4)[31,28],addr,00} | |
| 28 | JALR | PC=rs | |
| 29 | LB | LB rt, offset(base) | |
| 30 | LH | LH rt, offset(base) | |
| 31 | LW | LW rt, offset(base) | |
| 32 | SB | SB rt, offset(base) | |
| 33 | SH | SH rt, offset(base) | |
| 34 | SW | SW rt, offset(base) | |

表5.1-1 34条指令

### 数据通路

流水线的设计有多种想法，不过当然按照最经典的划分想法将执行一条指令的过程分成五个阶段：取指、译码、执行、访存和写回。

在上学期单周期CPU的设计中我们也是将指令按照这五个步骤来进行的，只是在单周期中并不存在多个指令在不同阶段的并行执行。而要将单周期CPU向多周期CPU进行转变就是要能让多条指令的不同阶段能够做到并行执：首先就是要将整个执行逻辑拆分开来，打断各个子模块之间的“路径”；但是其之间的信号流通没有改变，因而要在每两个子模块间添加一组流水线寄存器，通过将上一条指令的前阶段得出的东西存入流水线寄存器从而使得下一条指令不用等待上一条指令完全完成就可以进入并且还不会干扰上一条指令的执行；进一步地，针对并行执行指令出现的数据冒险、控制冒险，设计添加数据、控制信号的转发路径，根据控制信号从各阶段数据寄存器中选择正确的数据参与运算。

下面对每个阶段和每个阶段之间的流水线寄存器进行详细描述：

1. 取指阶段（IF\_stage）

该阶段的功能就是产生新的pc值和读取指令存储器，所以包含了单周期CPU中pc\_reg和instr\_memory两个子模块，因为instr\_memory之后要作为外部存储器，所以在这个阶段不加入。

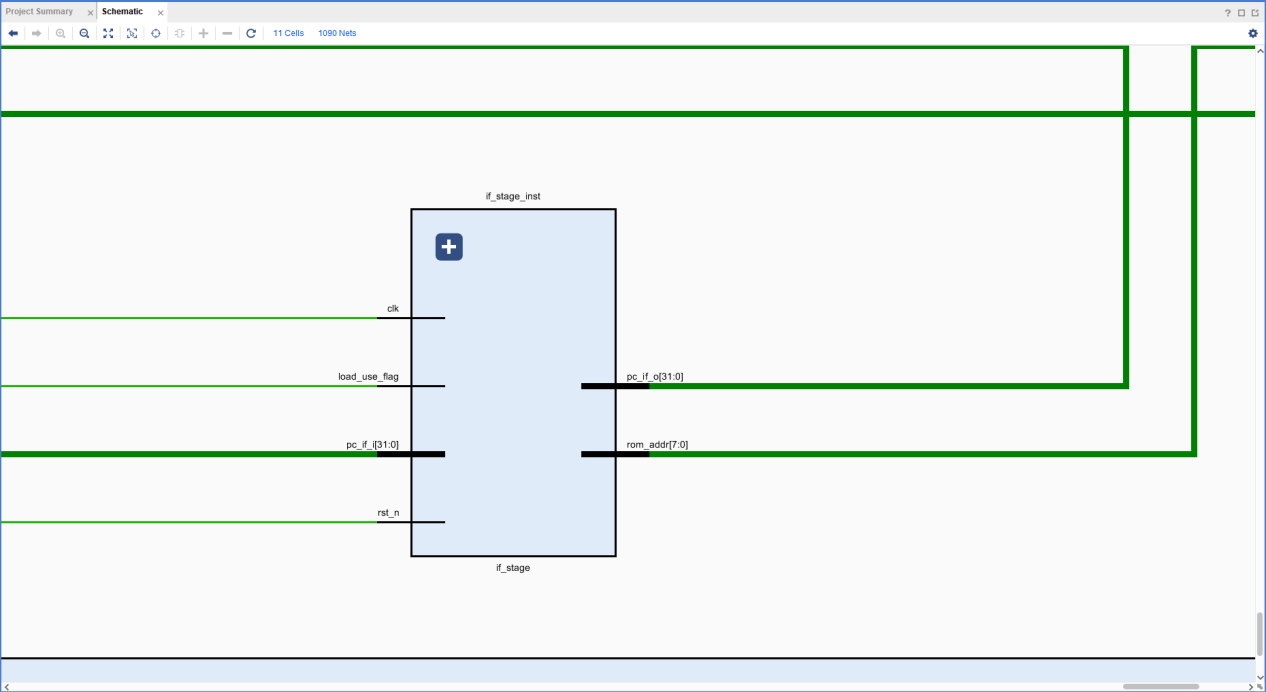


图5.1.1-1 取指阶段数据通路

1. IF\_ID流水线寄存器

在取指和译码阶段之间需要传递的信号是pc和读出的32位指令，因为在执行阶段计算下一个pc的值和跳转指令的目标地址时需要用到当前模块的pc所以pc需要逐级传递否则有可能被下一条指令更新丢失，保证pc和所在阶段执行的指令一致。

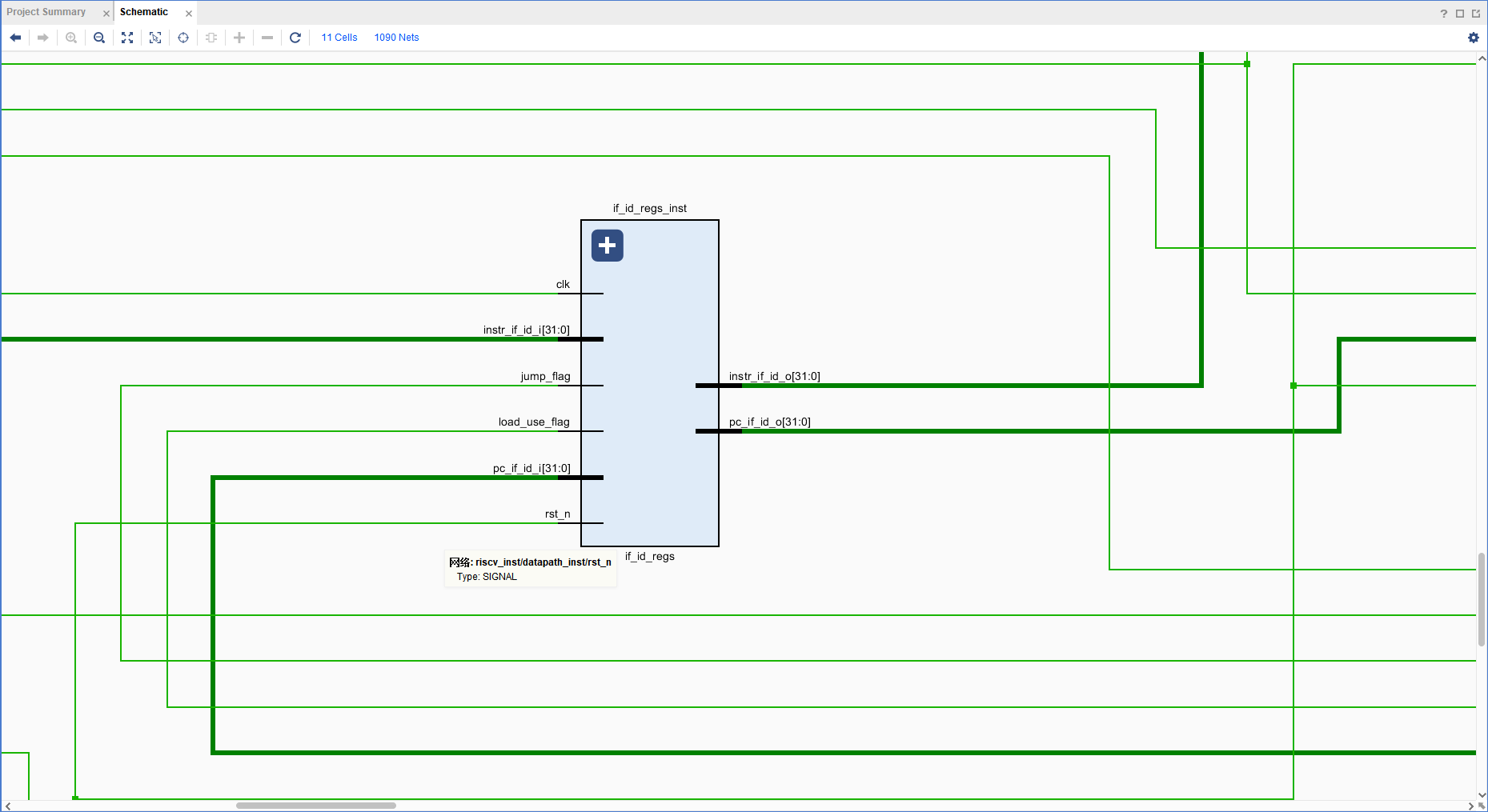


图5.1.1-2 IF\_ID流水线寄存器数据通路

1. 译码阶段（ID\_stage）

译码阶段要做的就是将传入的指令根据指令集进行译码然后读取寄存器堆，所以包含instr\_decode和registers两个模块，同时还要收到写回阶段传回来的rd信号，可以在此阶段将要写回寄存器的数据写回。

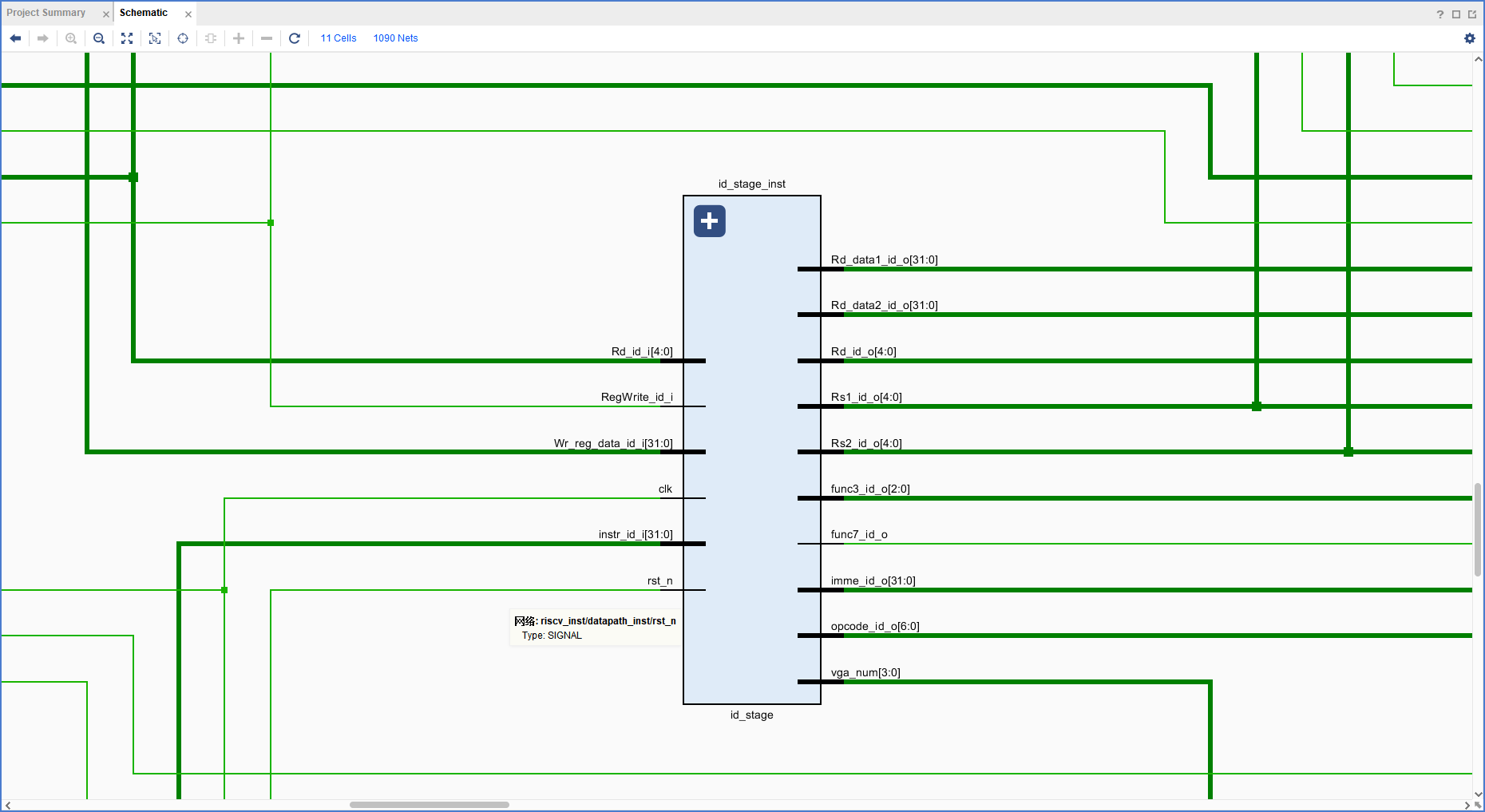


图5.1.1-3 译码阶段数据通路

1. ID\_EX流水线寄存器

在译码和执行阶段需要传递的信号首先还是需要有pc因为pc一直需要保存到最后写回阶段写回寄存器中，然后是译码阶段指令中包含的数据，可能是立即数也可能是寄存器堆读出来的数据。同样也需要传递rd信号直到写回阶段，但是可以不经过执行阶段。

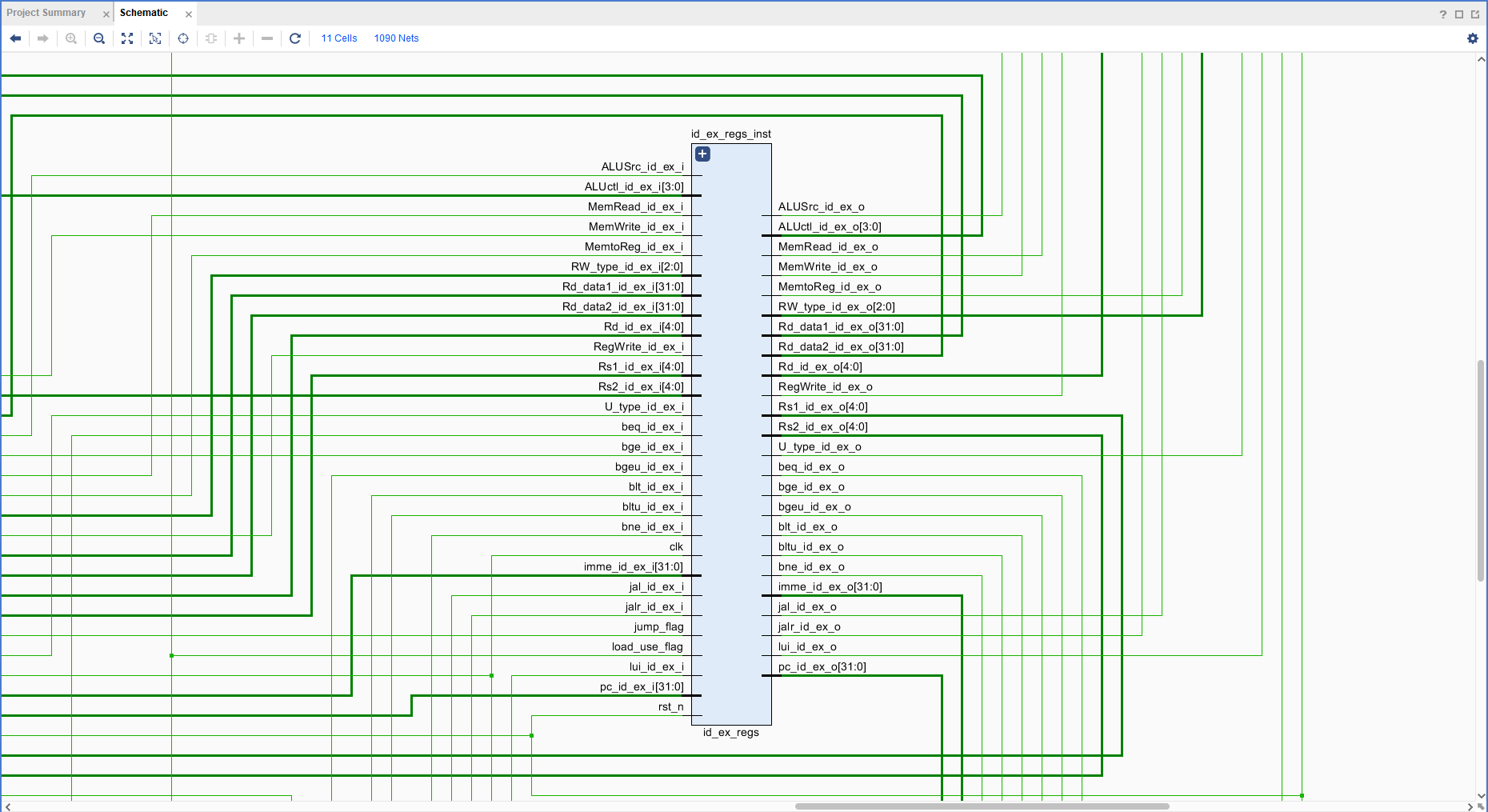


图5.1.1-4 4.ID\_EX流水线寄存器数据通路

1. 执行阶段（EX\_stage）

该阶段要做的事情就是alu的算术运算以及pc新的值的计算（包含得到跳转的地址值），可以说alu的结果一般是算术运算指令的结果、新的pc地址等，而得到什么结果取决于不同的数据来源也就是上个阶段传递过来的信号和控制器传递的控制信号。因而需要包含的是alu模块，branch\_judge模块，加法器模块（用于计算pc+4/imme），mux选择器模块（alu数据来源的选择，选择pc顺序执行或者pc跳转，jalr选择（即选择jalr还是跳转/顺序））。

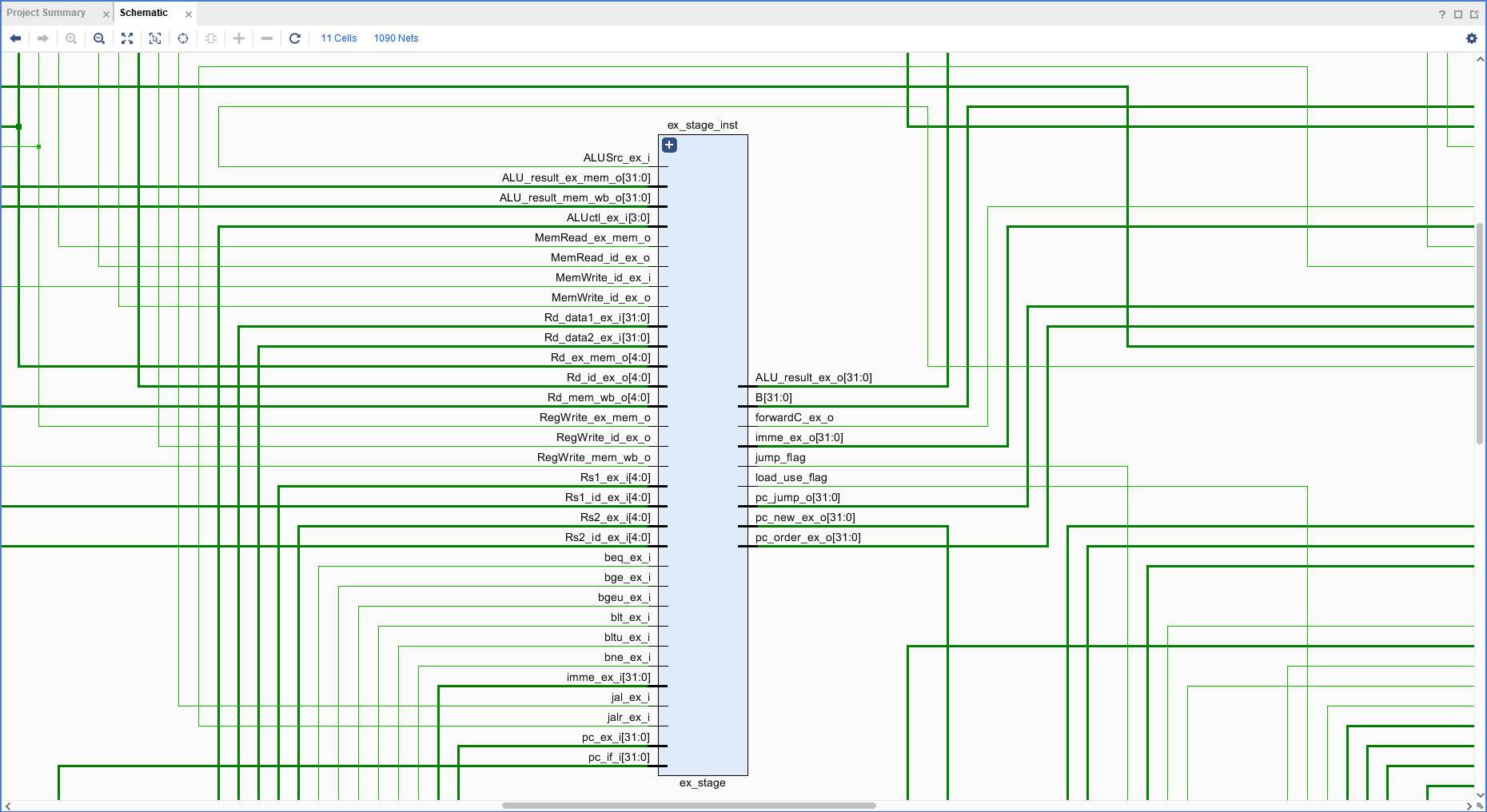


图5.1.1-5 执行阶段数据通路

1. EX\_MEM流水线寄存器

在执行阶段和访存阶段之间需要保存和传递的信号是ALU\_result,pc\_jump,pc\_order,Rd\_data2,imme,其中ALU\_result会在访存阶段作为访存对应的地址使用，在写回阶段作为运算结果写入写回寄存器，pc\_jump和pc\_order则是在写回阶段作为pc新值的待选写回寄存器，Rd\_data2则是在访存阶段作为写入存储器的数据使用，imme同样也在写回阶段使用，rd也需要继续传输。

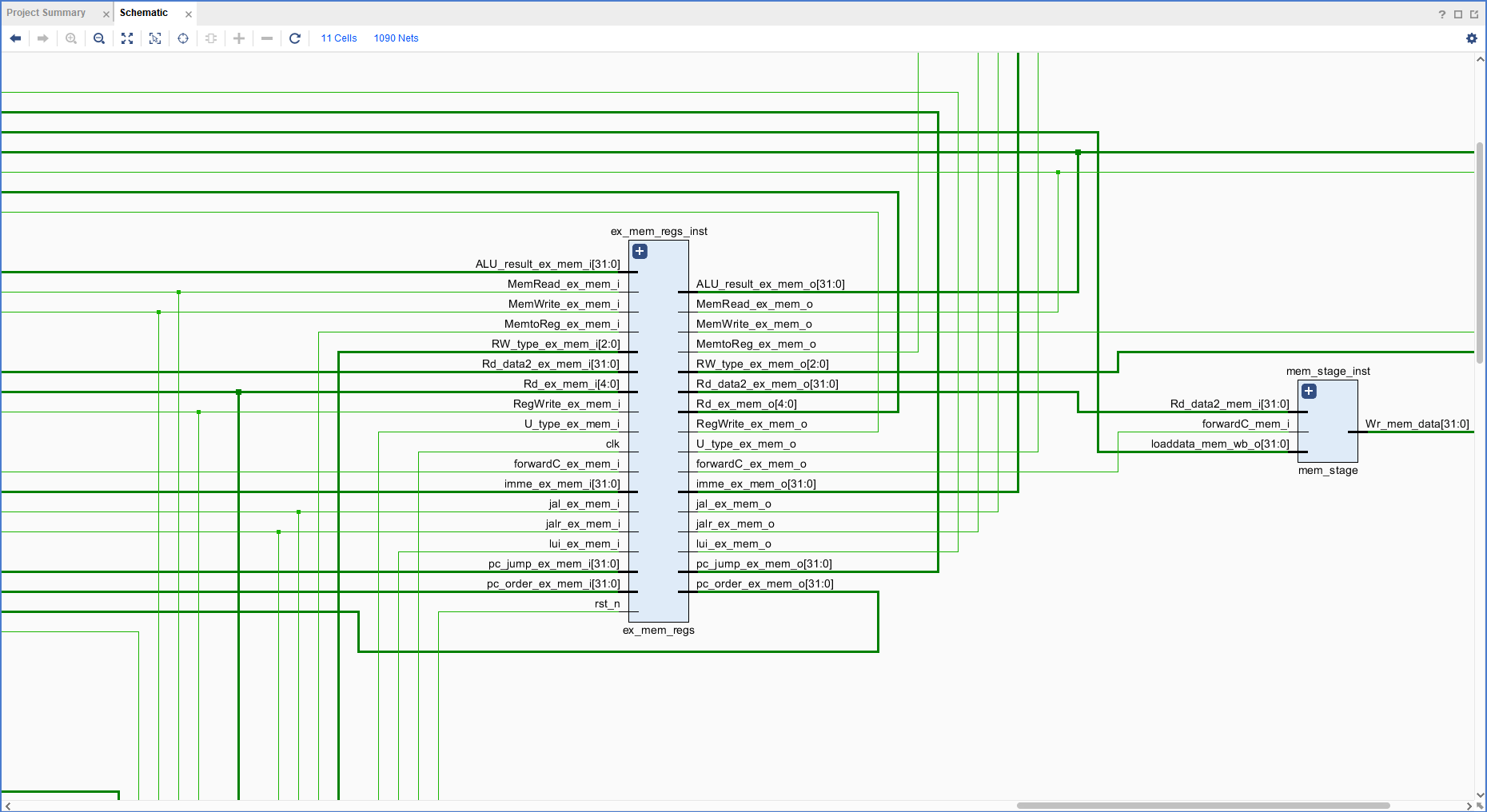


图5.1.1-6 6.EX\_MEM流水线寄存器数据通路

1. 访存阶段（MEM\_stage）

对于该阶段由于实验要求指令存储器和数据存储器都是和cpu分开的，因而只需要根据输出地址和数据给数据存储器，或者接收数据存储器读出的数据即可，直接在顶层的数据通路进行体现即可。

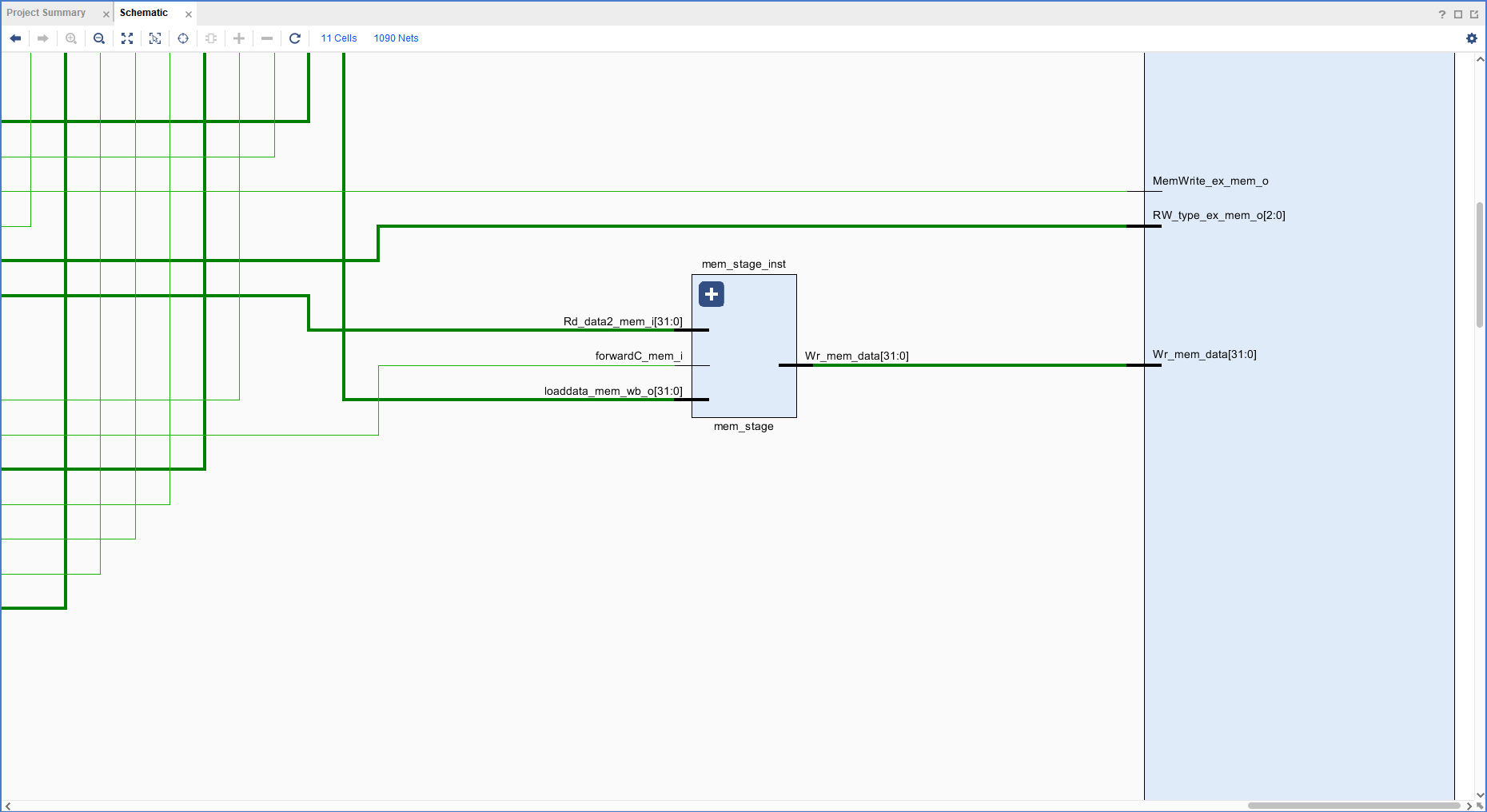


图5.1.1-7 访存阶段数据通路

1. MEM\_WB流水线寄存器

而从访存阶段到写回阶段所传递的需要使用的信号有ALU\_result,pc\_jump,pc\_order,rd，imme,loaddata,其中前面四种都是之前传递到写回阶段的信号，rd是直接传输到下一个周期的译码阶段直接找寄存器，而最后的loaddata就是在访存阶段读出来的数据，是新的需要传递的数据。

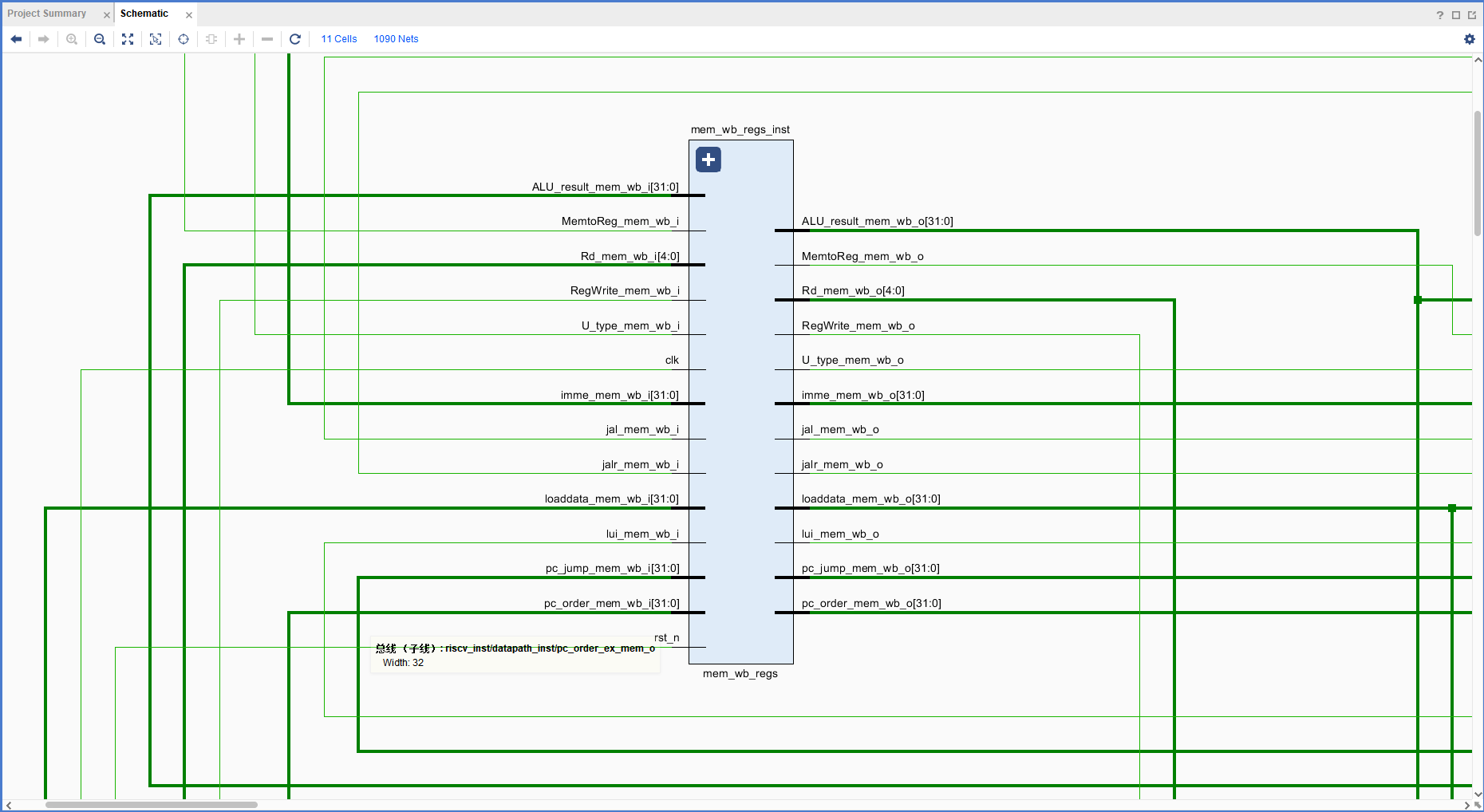


图5.1.1-8 8.MEM\_WB流水线寄存器数据通路

1. 写回阶段（WB\_stage）

写回阶段主要的任务就是通过四个选择器选择需要的数据写回寄存器堆（第一个是alu结果和读出的数据选择，然后是将其作为输入和传入的顺序运行pc进行选择而进行jalr选择，还有输入是顺序运行的pc和立即数跳转的lui选择，最后是将lui选择和jalr选择结果根据是否为u-type指令而选择出最后输出的一个数据）。

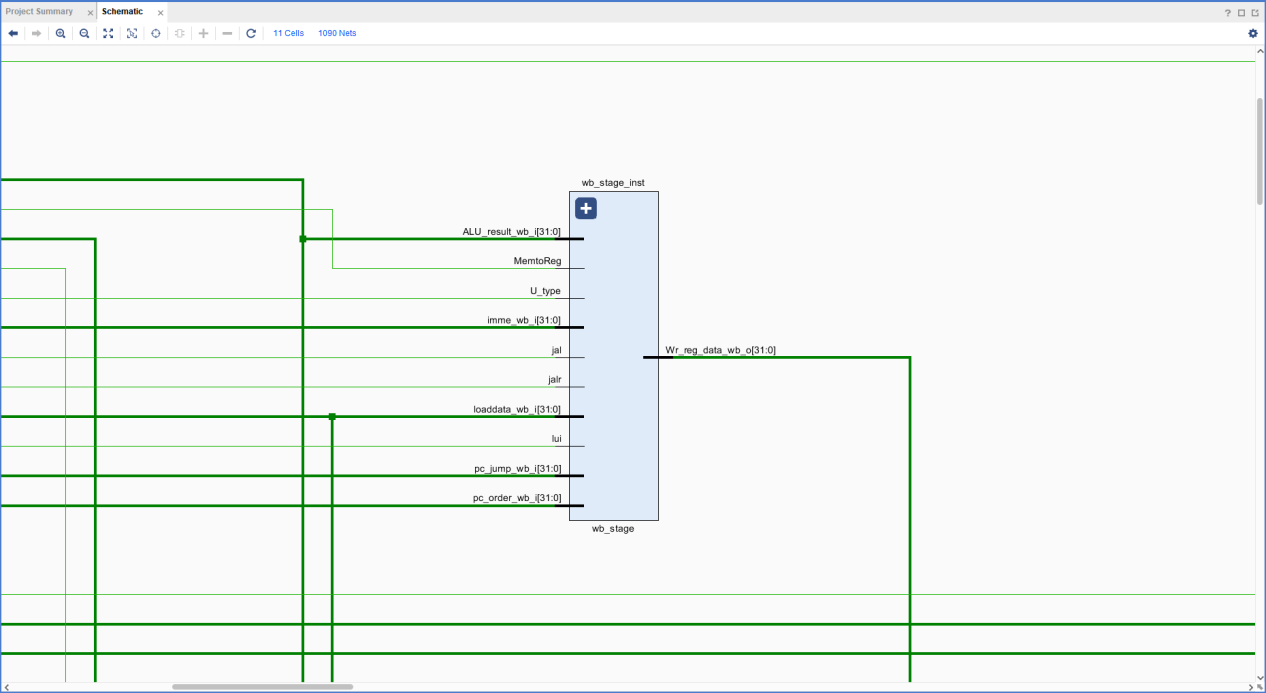


图5.1.1-9 写回阶段数据通路

1. 最后则是将流水线按顺序以及其中的流水线寄存器连接起来传递信号即可。

完成之上的步骤只是完成了基础的流水线架构，但是还有类比单周期CPU的控制器部分，也就是产生控制信号并填入数据通路中。

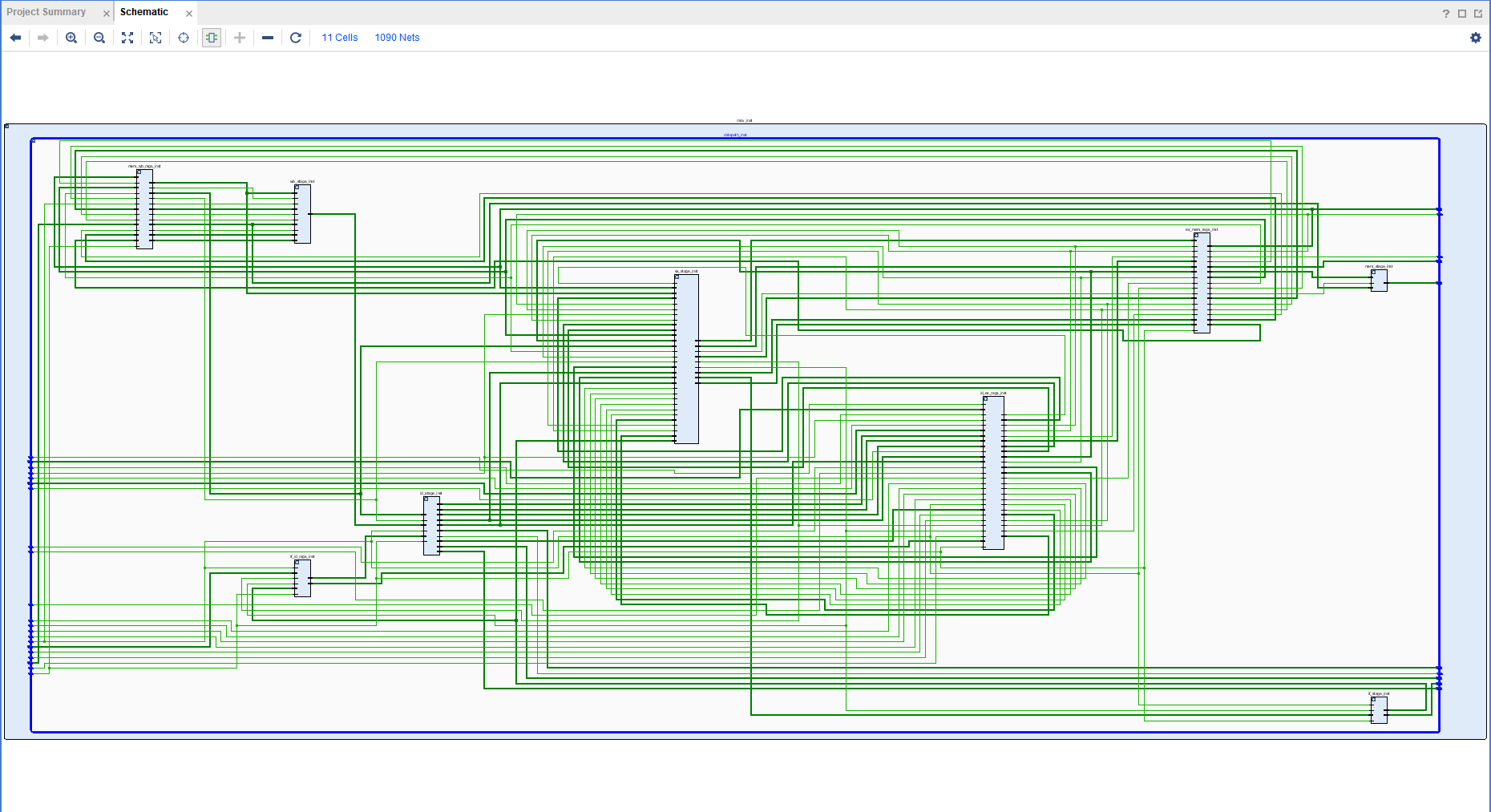


图5.1.1-10 5级流水线CPU数据通路

### 控制逻辑

流水线cpu的绝大部分控制信号和单周期cpu类似，采用组合逻辑硬联产生，控制器都是一样的，需要注意哪些控制信号需要随着流水线寄存器进行保存。

|  |  |  |  |
| --- | --- | --- | --- |
| **信号** | **位宽** | **输入/输出** | **描述** |
| Opcode | 7bit | Input | 操作码 |
| Func3 | 3bit | Input | 选择功能func3 |
| MemRead | 1bit | Output | 数据存储器读使能信号 |
| ALUop | 2bit | Output | ALU控制器的控制信号 |
| MemtoReg | 1bit | Output | 写回寄存器的数据类型控制信号 |
| MemWrite | 1bit | Output | 数据存储器写使能信号 |
| ALUSrc | 1bit | Output | ALU数据来源的数据类型控制信号 |
| RegWrite | 1bit | Output | 寄存器的写使能控制信号 |
| lui | 1bit | Output | 写回寄存器的数据类型控制信号 |
| U\_type | 1bit | Output | 写回寄存器的数据类型控制信号 |
| jal | 1bit | Output | 选择pc的类型的控制信号以及写回寄存器的数据类型的控制信号 |
| jalr | 1bit | Output | 选择pc的类型的控制信号以及写回寄存器的数据类型的控制信号 |
| beq | 1bit | Output | 判断是否跳转的控制信号 |
| bne | 1bit | Output | 判断是否跳转的控制信号 |
| blt | 1bit | Output | 判断是否跳转的控制信号 |
| bge | 1bit | Output | 判断是否跳转的控制信号 |
| bltu | 1bit | Output | 判断是否跳转的控制信号 |
| bgeu | 1bit | Output | 判断是否跳转的控制信号 |
| RW\_type | 3bit | Output | 数据存储器的读（写）选择 |

需要注意的是控制信号的保存规律，首先所有控制信号都在id阶段进行产生，接下来按顺序将控制信号一层层使用过然后不进行保存。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | ID/EX | EX | EX/MEM | MEM | MEM/WB | WB |
| ALUSrc | ALUSrc |  |  |  |  |  |
| ALUctl | ALUctl |  |  |  |  |  |
| Beq | Beq |  |  |  |  |  |
| Bne | Bne |  |  |  |  |  |
| Blt | Blt |  |  |  |  |  |
| Bge | Bge |  |  |  |  |  |
| Bltu | Bltu |  |  |  |  |  |
| Bgeu | Bgeu |  |  |  |  |  |
| Jal | Jal | Jal | Jal | Jal | Jal |  |
| Jalr | Jalr | Jalr | Jalr | Jalr | Jalr |  |
| MemRead | MemRead | MemRead | MemRead |  |  |  |
| MemWrite | MemWrite | MemWrite | MemWrite |  |  |  |
| RW\_type | RW\_type | RW\_type | RW\_type |  |  |  |
| Lui | Lui | Lui | Lui | Lui | Lui |  |
| U\_type | U\_type | U\_type | U\_type | U\_type | U\_type |  |
| MemtoReg | MemtoReg | MemtoReg | MemtoReg | MemtoReg | MemtoReg |  |
| RegWrite | RegWrite | RegWrite | RegWrite | RegWrite | RegWrite | RegWrite |

信号的作用描述

|  |  |  |  |
| --- | --- | --- | --- |
| **信号** | **位宽** | **输入/输出** | **描述** |
| Opcode | 7bit | Input | 操作码 |
| Func3 | 3bit | Input | 选择功能func3 |
| MemRead | 1bit | Output | 数据存储器读使能信号 |
| ALUctl | 2bit | Output | ALU控制器的控制信号 |
| MemtoReg | 1bit | Output | 写回寄存器的数据类型控制信号 |
| MemWrite | 1bit | Output | 数据存储器写使能信号 |
| ALUSrc | 1bit | Output | ALU数据来源的数据类型控制信号 |
| RegWrite | 1bit | Output | 寄存器的写使能控制信号 |
| lui | 1bit | Output | 写回寄存器的数据类型控制信号 |
| U\_type | 1bit | Output | 写回寄存器的数据类型控制信号 |
| jal | 1bit | Output | 选择pc的类型的控制信号以及写回寄存器的数据类型的控制信号 |
| jalr | 1bit | Output | 选择pc的类型的控制信号以及写回寄存器的数据类型的控制信号 |
| beq | 1bit | Output | 判断是否跳转的控制信号 |
| bne | 1bit | Output | 判断是否跳转的控制信号 |
| blt | 1bit | Output | 判断是否跳转的控制信号 |
| bge | 1bit | Output | 判断是否跳转的控制信号 |
| bltu | 1bit | Output | 判断是否跳转的控制信号 |
| bgeu | 1bit | Output | 判断是否跳转的控制信号 |
| RW\_type | 3bit | Output | 数据存储器的读（写）选择 |

流水线cpu相对于单周期cpu新增了数据冒险和控制冒险，需要新增控制信号：forwardA、forwardB、forwardC、load\_use\_flag；

1. forwardA（forwardB）信号：选择ALU操作数A（B）从上一阶段寄存器获得还是从ex-mem阶段寄、mem-wb阶段寄存器转发。
2. forwardC信号：选择访存阶段的写入数据是从上一阶段的寄存器获得还是从上一条指令的mem-wb阶段寄存器转发。
3. load\_use\_flag信号：出现load-use类型冒险时，控制流水线暂停。

## 流水线冒险问题以及解决方案

1. 数据冒险

数据冒险出现的主要原因是：由于流水线cpu并发指令多条指令，可能导致流水线上后一条指令需要使用前一条指令的运算结果；共4种类型。

* 1. 在一个周期开始，EX 阶段要使用上一条处在 EX 阶段指令的执行结果，此时需要将 EX/MEM 寄存器的数据前递。
  2. 在一个周期开始，EX 阶段要使用上一条处在 MEM 阶段指令的执行结果，此时我们将 MEM/WB 寄存器的数据前递。
  3. 在一个周期开始，EX 阶段要使用上一条处在 WB 阶段指令的执行结果，此时使用寄存器堆前递机制（即在clk信号下降沿写入寄存器）。
  4. 在发生load-use型冒险的时候，如果是load后跟着store指令，并且load指令的rd与store指令的rs1 不同而与rs2相同，此时为load-use的特殊情形：不需要停顿流水线，而将MEM/WB 寄存器的数据前递到MEM阶段。

1. 控制冒险

控制冒险指流水线CPU在处理条件分支指令时，由于跳转条件是否成立需要在执行阶段才能确定，而流水线会不断流入后续指令，导致出现流入流水线的后续指令并非程序所需要的指令的情况。出现控制冒险时需要对流水线进行停顿或者冲刷，来处理流入流水线的多余指令。

在本CPU中，在发生控制冒险时采取冲刷后续两条指令的方式进行解决。在执行阶段设置生成控制信号判断控制冒险的发生，若发生控制冒险，则将IF\_ID流水线寄存器和ID\_EX流水线寄存器中的数据清空，即实现了对当前处于执行阶段指令的后续两条指令的冲刷。

# 设计与实现

具体设计的思路已经在数据通路阶段进行了详细说明。

设计并实现了以下模块：

1. riscv\_top：顶层模块，负责连接cpu和指令和数据两个存储器进行数据交换
2. instr\_mem和data\_mem：分别作为指令存储器和内存存储数据
3. control和datapath：控制模块，进行控制信号的产生以及数据通路，控制信号传递给数据通路，数据通路内进行cpu各个阶段之间的数据传输，包含各阶段的模块和流水线寄存器以及各种前递和暂停单元。
4. if\_stage:取指阶段，根据pc将对应的指令从指令存储器中取出，包含pc\_reg模块
5. id\_stage:译码阶段，将取出的指令进行译码，得到控制信号和指令中的数据，指引指令的执行方式。包含instr\_decode译码模块和registers寄存器模块。
6. ex\_stage:执行阶段，进行alu计算，跳转指令执行pc计算，计算指令执行逻辑、移位等运算，根据计算结果判定分支，包含分支判断、alu模块
7. mem\_stage:访存阶段，与外部存储器进行数据交换的阶段
8. wb\_stage:写回阶段，选择要写回的数据
9. 每个阶段间的流水线寄存器；该模块对需要保存的信号进行流水线保存，从而保证时序正常，既能读入新的指令又能保证结果不出现错误。

# 测试

## 仿真测试用例、结果以及分析

## 7.1 功能性测试

7.1.1数据冒险测试：

测试用例代码1：

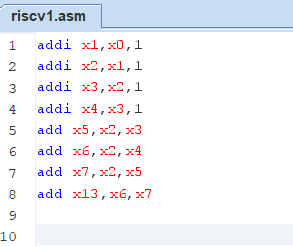


图7.1.1-1 测试用例代码1

本测试用例用于测试前三种数据冒险，预期结果如下表：

|  |  |
| --- | --- |
| 寄存器 | 值 |
| x1 | 1 |
| x2 | 2 |
| x3 | 3 |
| x4 | 4 |
| x5 | 5 |
| x6 | 6 |
| x7 | 7 |
| x13 | 13 |

仿真结果1：

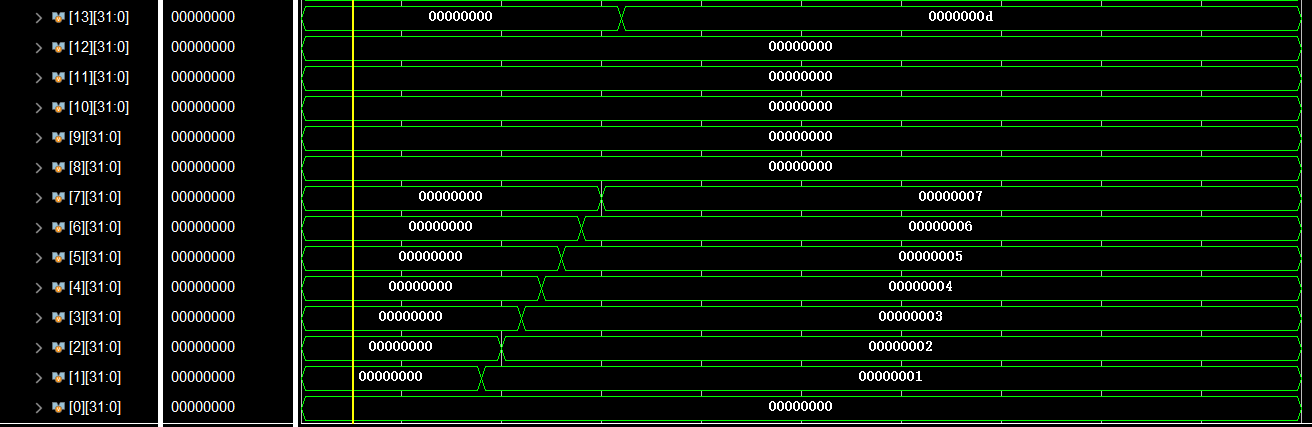


图7.1.1-2 仿真结果1

符合预期结果，说明CPU能处理前三种数据冒险。

测试用例代码2：

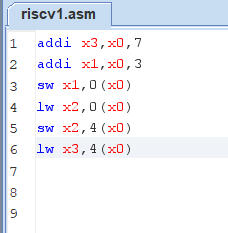


图7.1.1-3 测试用例代码2

本测试用例用于测试第四种种数据冒险，预期结果如下表：

|  |  |
| --- | --- |
| 寄存器/内存 | 值 |
| x1 | 3 |
| x2 | 3 |
| x3 | 7→3 |
| ram[0] | 3 |
| ram[1] | 3 |

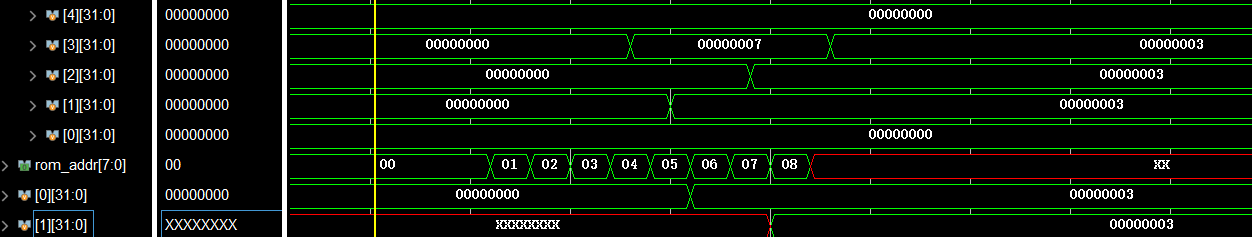
仿真结果2：

图7.1.1-4 仿真结果2

符合预期结果，说明CPU能处理前三种数据冒险。

测试用例代码3：

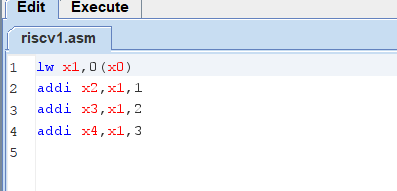


图7.1.1-5 测试用例代码3

本测试用例用于测试第四种种数据冒险，预期结果如下表：

|  |  |
| --- | --- |
| 寄存器 | 值 |
| x1 | 0 |
| x2 | 1 |
| x3 | 2 |
| x4 | 3 |

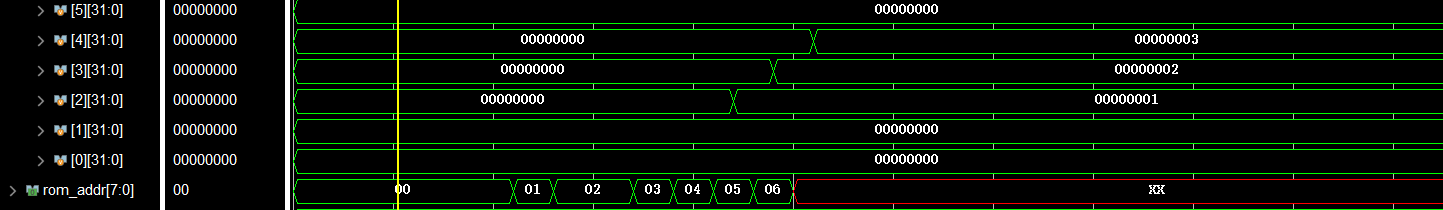
仿真结果3：

图7.1.1-6 仿真结果3

符合预期结果，说明CPU能处理第五种数据冒险（加载——使用型）。

7.1.2数据冒险测试：

测试用例代码：

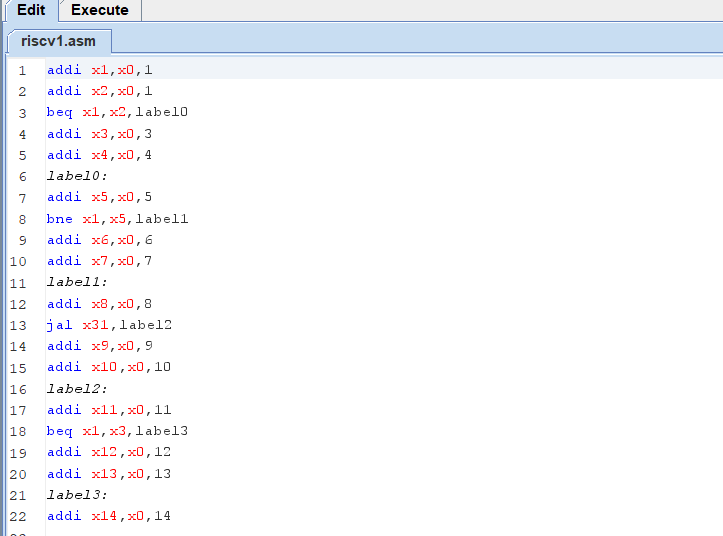


图7.1.2-1 测试用例代码4

本测试用例用于测试第四种种数据冒险，预期结果如下表：

|  |  |
| --- | --- |
| 寄存器 | 值 |
| x1 | 1 |
| x2 | 1 |
| x3 | 0 |
| x4 | 0 |
| x5 | 5 |
| x6 | 0 |
| x7 | 0 |
| x8 | 8 |
| x9 | 0 |
| x10 | 0 |
| x11 | 11 |
| x12 | 12 |
| x13 | 13 |
| x14 | 14 |

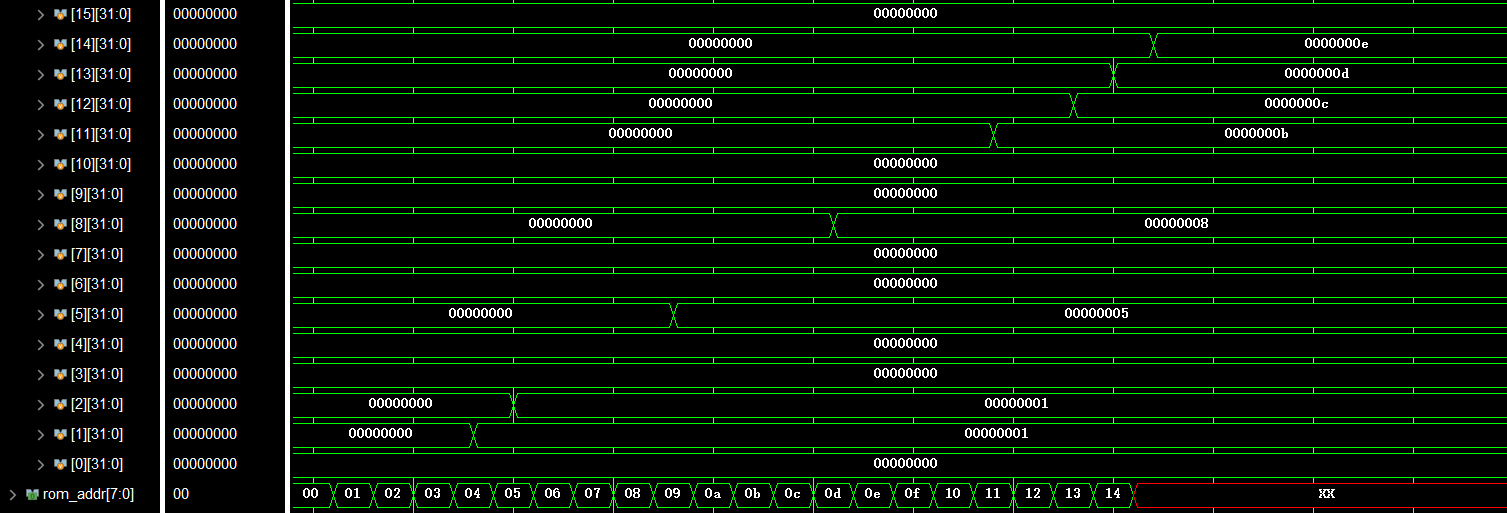
仿真结果：

图7.1.2-2 仿真结果4

符合预期结果，说明CPU能正常冲刷条件跳转指令后的两条指令，能处理控制冒险。

## 7.2性能测试

测试用例代码：

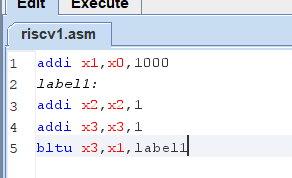


图7.2-1 测试用例代码5

本测试用例用于测试CPU对超长循环的处理能力，运行100ns。

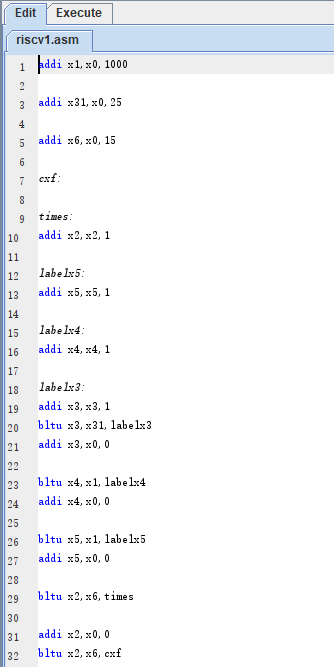
仿真结果：

图7.2-2 仿真结果5

在运行时间内CPU持续并正确运行指令，性能良好。

## 7.3下板测试

下板测试选择使用实验设计cpu控制外设显示15秒倒计时。为了更好测试cpu执行性能，测试使用多层嵌套循环代替中断-阻塞方式控制外设，代码实现如下：



上板实操流程为：使用mars软件将risc-v代码dump为二进制；修改.coe文件并将指令寄存器封装为IP核；综合流水线cpu代码、外设控制代码、指令寄存器；将综合结果转化为bitstream并写入设备。

经过参数调试，计时器准确同步常数级的计算开销，实现了15秒的倒计时。



图7.3-1 上板测试结果

# 问题及解决方法

在进行对数据冒险和控制冒险的改动时，团队在添加前递信号后对前递的数据在数据通路的流向上收获了许多新的发现：

在数据冒险部分，我们首先在EX阶段产生的ALU计算结果ALU\_result的前递位置上产生了错误判断，在出现数据冲突时需要将EX\_MEM和MEM\_WB寄存器中的ALU\_result信号前递到EX阶段，进行前递信号的生成和选择。在检查仿真信号和推导流水线CPU的时序后找到问题，并进行解决。

在ALU操作数的选择上，应该先使用前递选择信号forwardB对前递回的ALU\_result数据和译码读取出的Rd\_data2数据进行选择，然后再将选择的结果与读取出的立即数imme进行选择，将最后的结果传递给ALU进行计算。我们在设计初期错误地判断了两次选择的顺序，导致ALU在处理含立即数的命令时会出错。经过调试和对CPU结构进行分析后找到问题根源并解决。

在处理第三种数据冒险时，forwardC应该在EX阶段产生后在EX\_MEM寄存器中保存一个时钟周期再传递给MEM阶段进行写入数据的选择，我们在设计初期忽略了forwardC的保存，将其直接传递到MEM阶段，导致CPU在内存写入时出现了问题。通过对流水线CPU的信号梳理，我们推导出了forwardC信号需要保存的路径，解决了问题。

在使用jal命令时，我们发现CPU在设计时在X1寄存器存储跳转地址，若在此后使用X1寄存器的内容，会发现原先存储在X1中的数据发生了改变，造成程序出错。因此在使用jal命令时选择使用其他寄存器代替。

# 心得体会及总结

通过本次流水线CPU的团队设计，我们团队不仅提升了交流解决问题合作开发的能力，还加深巩固了上学期计算机组成原理课程所学习的理论知识，并将其用于实践中，同时解决流水线的时序问题加深了我们对CPU数据通路的理解，在解决时序问题中学会了用列表的方式更有条理的整理每一个周期各个指令的执行情况。

# 参考文献有价值的资源推荐