****

**东北大学秦皇岛分校**

**计算机与通信工程学院**

**计算机组成原理课程设计**

**设计题目** 数字跑表模块和流水线CPU的设计与实现

|  |  |
| --- | --- |
| 专业名称 | 计算机科学与技术 |
| 班级学号 | 计科2205-31-202212701 |
| 学生姓名 | 陈宇帆 |
| 指导教师 | 沈哲 |
| 设计时间 | **2025年1月6日—2025年1月12日** |

**课程设计任务书**

**专业：**计算机科学与技术 **学号：**202212701 **学生姓名（签名）：**

**设计题目：** 数字跑表模块与流水线CPU的设计与实现

**一、设计实验条件**

计算机与通信工程学院实验室，综合楼1207

**二、设计任务及要求**

1. **电路模块设计**：用Verilog实现一个电路模块；
2. **流水线CPU设计**：至少实现21条指令ADD、SUB、OR、AND、XOR、NOR、SLT、SLTU、SLL、SRL、SRA、LUI， ADDU、ADDIU、SUBU、LW、SW、BEQ、BNE、JAL、JR。

要求如下：

1. CPU微结构为静态5级流水。
2. 实现MIPS架构的延迟槽技术，延迟槽可以是任意指令。
3. 控制相关由分支指令造成，通过延迟槽技术可以完美解决。
4. 结构相关即某一级流水停顿了，会阻塞上游的流水级。
5. 要求仿真运行测试程序通过。

**三、设计报告的内容**

1. **设计题目与设计任务（设计任务书）**
2. **前言（绪论）(设计的目的、意义等)**
3. **设计主体（各部分设计内容、分析、结论等）**
4. **结束语（设计的收获、体会等）**
5. **参考资料**

**四、设计时间与安排**

**1、设计时间： 1周**

**2、设计时间安排：**

熟悉实验设备、收集资料： 1 天

设计图纸、实验、计算、程序编写调试： 4 天

编写课程设计报告： 1 天

答辩： 1 天

# 目录

[目录 3](#_Toc187415507)

[【课程设计内容】 5](#_Toc187415508)

[【系统设计与实现】 6](#_Toc187415509)

[1、 数字跑表电路模块设计 6](#_Toc187415510)

[1.1 数字跑表电路模块介绍 6](#_Toc187415511)

[1.2 数字跑表电路模块原理 6](#_Toc187415512)

[1.3 数字跑表电路模块的顶层设计原理图 6](#_Toc187415513)

[1.4 数字跑表电路模块的关键代码 6](#_Toc187415514)

[1.5 数字跑表电路模块的综合结果 6](#_Toc187415515)

[2、 五级流水CPU介绍 7](#_Toc187415516)

[2.1 实现的指令列表（包括指令描述） 7](#_Toc187415517)

[2.2 流水线数据流图 14](#_Toc187415518)

[2.3 流水线模块结构 14](#_Toc187415519)

[2.4 流水线模块接口描述（包括关键代码） 20](#_Toc187415520)

[2.5 综合结果 43](#_Toc187415521)

[【系统测试】 44](#_Toc187415522)

[1、 数字跑表系统测试 44](#_Toc187415523)

[1. 数字跑表的功能仿真 44](#_Toc187415524)

[2. 数字跑表的RTL级逻辑电路 44](#_Toc187415525)

[五级流水CPU系统测试 44](#_Toc187415526)

[1. 五级流水CPU的功能仿真 44](#_Toc187415527)

[2. 五级流水CPU的RTL级逻辑电路 44](#_Toc187415528)

[【总结】 45](#_Toc187415529)

[【心得体会】 46](#_Toc187415530)

[【参考文献（资料）】 47](#_Toc187415531)

# 【课程设计内容】

**题目**：数字跑表和MIPS32五级流水CPU设计

**设计内容如下**：

1. 电路模块设计：用Verilog实现一个数字跑表电路模块；

2. 流水线CPU设计：至少实现21条指令ADD、SUB、OR、AND、XOR、NOR、SLT、SLTU、SLL、SRL、SRA、LUI， ADDU、ADDIU、SUBU、LW、SW、BEQ、BNE、JAL、JR。

要求如下：

（1） CPU微结构为静态5级流水。

（2） 实现MIPS架构的延迟槽技术，延迟槽可以是任意指令。

（3） 控制相关由分支指令造成，通过延迟槽技术可以完美解决。

（4） 结构相关即某一级流水停顿了，会阻塞上游的流水级。

（5） 要求仿真运行测试程序通过。

# 【系统设计与实现】

## 1、 数字跑表电路模块设计

### 1.1 数字跑表电路模块介绍

使用Verilog HDL设计一个数字跑表，具有复位、暂停、秒表等功能，编写测试代码进行仿真验证。

### 1.2 数字跑表电路模块原理

输入系统时钟CLK为50MHZ。我们要做的就是通过计算经过多少个系统的CLK时钟，得到一个毫秒（百分秒）的输出，让它进一步作为秒、分钟的输入时钟信号。例如：通过100个毫秒的输入得到1个1秒的输出，再通过60个1秒得到1个1分钟的输出。

### 1.3 数字跑表电路模块的顶层设计原理图

### 1.4 数字跑表电路模块的关键代码

从题目分析我们看到：

1、我们需要一个将系统内部时钟转换为一个毫秒信号时钟的分频模块。

2、需要将输入毫秒ms时钟信号转化能输出秒s，分钟m的模块。

3、需要能将毫秒ms、秒s、分钟m在数码管上显示的模块。

4、最后通过顶层模块top将所有模块串联起来实现功能。

### 1.5 数字跑表电路模块的综合结果

## 2、 五级流水CPU介绍

### 2.1 实现的指令列表（包括指令描述）

处理器需要实现的指令包括所有非浮点 MIPS I 指令以及 MIPS32 中的 ERET 指令，有 14 条算术运算指令、8条逻辑运算指令，6 条移位指令、12 条分支跳转指令、4 条数据移动指令、2 条自陷指令、12 条访存指令、3 条特权指令，共计 61 条。下面分类给出各部分指令的简要功能介绍。

表 1 算术运算指令

|  |  |
| --- | --- |
| **指令名称格式** | **指令功能简述** |
| ADD rd, rs, rt | 加（可产生溢出例外） |
| ADDI rt, rs, immediate | 加立即数（可产生溢出例外） |
| ADDU rd, rs, rt | 加（不产生溢出例外） |
| ADDIU rt, rs, immeidate | 加立即数（不产生溢出例外） |
| SUB rd. rs, rt | 减（可产生溢出例外） |
| SUBU rd, rs, rt | 减（不产生溢出例外） |
| SLT rd, rs, rt | 有符号小于置 1 |
| SLTI rt, rs, immediate | 有符号小于立即数设置 1 |
| SLTU rd, rs, rt | 无符号小于设置 1 |
| SLTIU rt, rs, immediate | 无符号小于立即数设置 1 |
| DIV rs, rt | 有符号字除 |
| DIVU rs,rt | 无符号字除 |
| MULT rs, rt | 有符号字乘 |
| MULTU rs, rt | 无符号字乘 |

表 2 逻辑运算指令

|  |  |
| --- | --- |
| **指令名称格式** | **指令功能简述** |
| AND rd, rs, rt | 位与 |
| ANDI rt, rs, immediate | 立即数位与 |
| LUI rt,immediate | 寄存器高半部分置立即数 |
| NOR rd, rs, rt | 位或非 |
| OR rd, rs, rt | 位或 |
| ORI rt, rs, immediate | 立即数位或 |
| XOR rd, rs, rt | 位异或 |
| XORI rt, rs, immediate | 立即数位异或 |

表 3 移位指令

|  |  |
| --- | --- |
| **指令名称格式** | **指令功能简述** |
| SLL rd, rt, sa | 立即数逻辑左移 |
| SLLV rd, rs, rt | 变量逻辑左移 |
| SRA rd, rt, sa | 立即数算术右移 |
| SRAV rd, rs, rt | 变量算术右移 |
| SRL rd, rt, sa | 立即数逻辑右移 |
| SRLV rd, rs, rt | 变量逻辑右移 |

表 4 分支跳转指令

|  |  |
| --- | --- |
| **指令名称格式** | **指令功能简述** |
| BEQ rs, rt, offset | 相等转移 |
| BNE rs, rt, offset | 不等转移 |
| BGEZ rs, offset | 大于等于 0 转移 |
| BGTZ rs, offset | 大于 0 转移 |
| BLEZ rs, offset | 小于等于 0 转移 |
| BLTZ rs, offset | 小于 0 转移 |
| BLTZAL rs, offset | 小于 0 调用子程序并保存返回地址 |
| BGEZAL rs, offset | 大于等于 0 调用子程序并保存返回地址 |
| J target | 无条件直接跳转 |
| JAL target | 无条件直接跳转至子程序并保存返回地址 |
| JR rs | 无条件寄存器跳转 |
| JALR rd, rs | 无条件寄存器跳转至子程序并保存返回地址下 |

表 5 数据移动指令

|  |  |
| --- | --- |
| **指令名称格式** | **指令功能简述** |
| MFHI rd | HI 寄存器至通用寄存器 |
| MFLO rd | LO 寄存器至通用寄存器 |
| MTHI rs | 通用寄存器至 HI 寄存器 |
| MTLO rs | 通用寄存器至 LO 寄存器 |

表 6 自陷指令

|  |  |
| --- | --- |
| **指令名称格式** | **指令功能简述** |
| BREAK | 断点 |
| SYSCALL | 系统调用 |

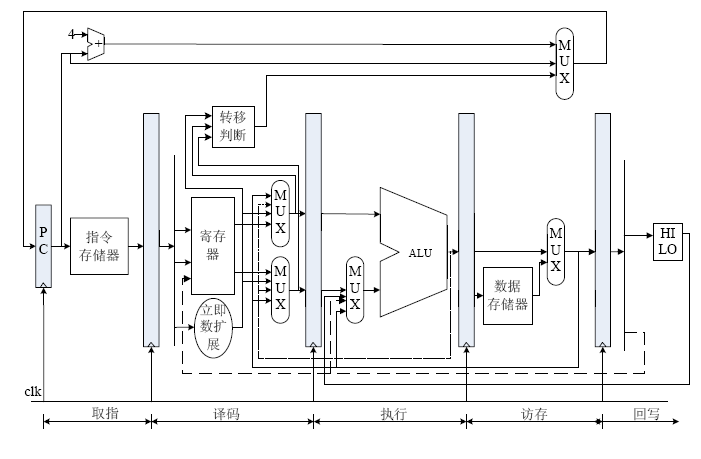
表 7 访存指令

|  |  |
| --- | --- |
| **指令名称格式** | **指令功能简述** |
| LB rt, offset(base) | 取字节有符号扩展 |
| LBU rt, offset(base) | 取字节无符号扩展 |
| LH rt, offset(base) | 取半字有符号扩展 |
| LHU rt, offset(base) | 取半字无符号扩展 |
| LW rt, offset(base) | 取字 |
| LWL rt, offset(base) | 非对齐地址取字至寄存器左部 |
| LWR rt, offset(base) | 非对齐地址取字至寄存器右部 |
| SB rt, offset(base) | 存字节 |
| SH rt, offset(base) | 存半字 |
| SW rt, offset(base) | 存字 |
| SWL rt, offset(base) | 寄存器左部存入非对齐地址 |
| SWR rt, offset(base) | 寄存器右部存入非对齐地址 |

表 8 特权指令

|  |  |
| --- | --- |
| **指令名称格式** | **指令功能简述** |
| ERET | 例外处理返回 |
| MFC0 | 读 CP0 寄存器值至通用寄存器 |
| MTC0 | 通用寄存器值写入 CP0 寄存器 |

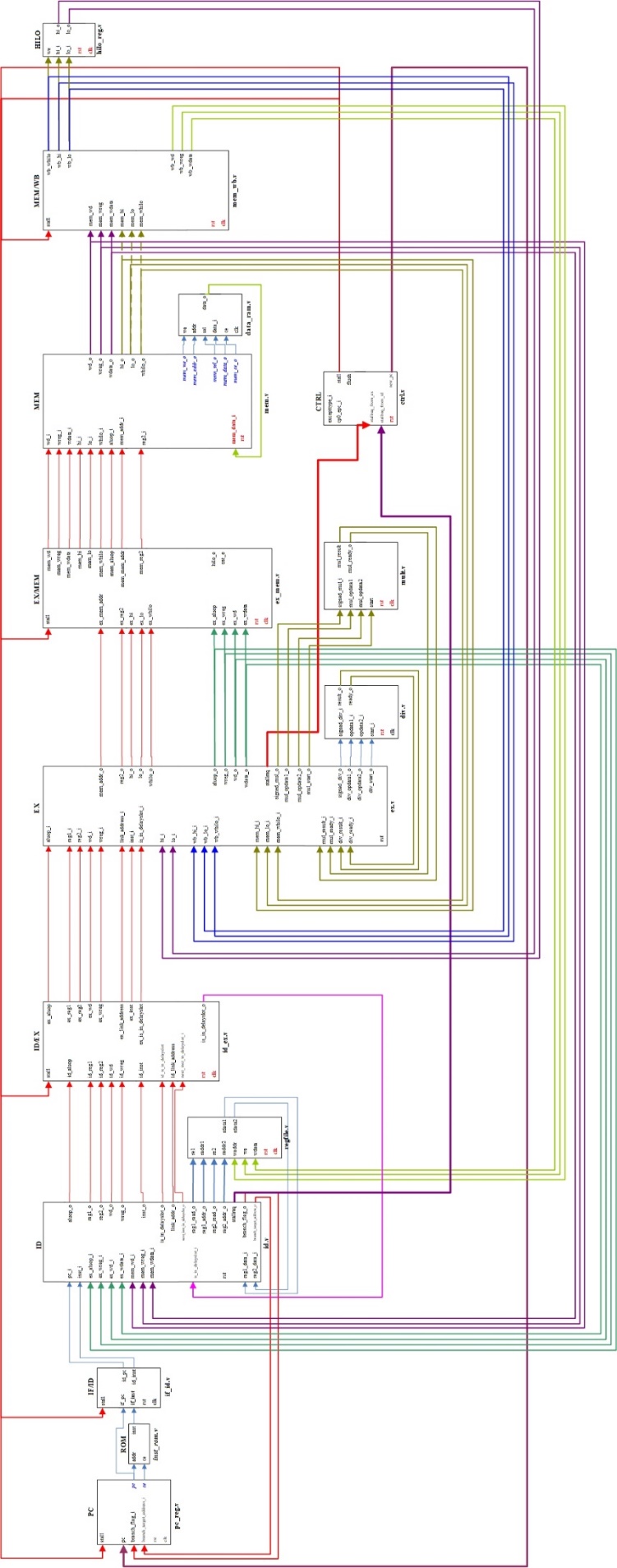
### 2.2 流水线数据流图



### 2.3 流水线模块结构

#### 2.3.1 五级流水CPU系统结构图

图 五级流水CPU系统结构图



#### 2.3.2 各个模块的具体结构图

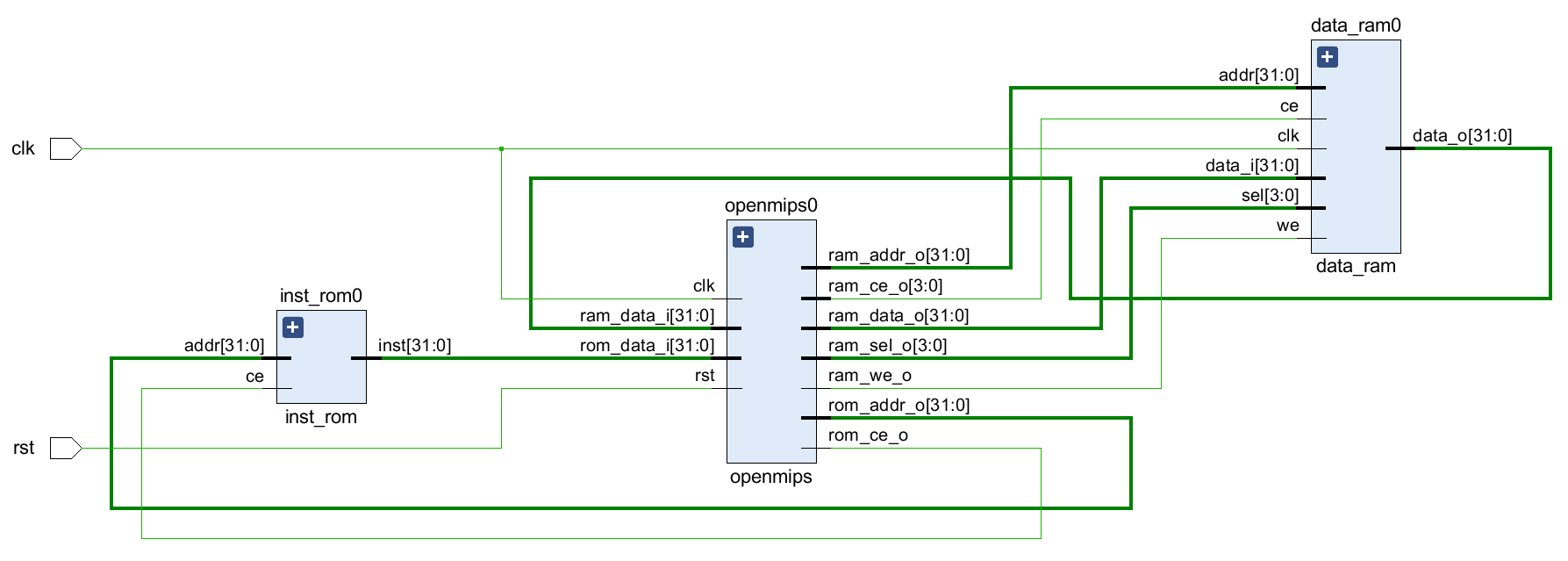


图 Openmips\_min\_sopc模块

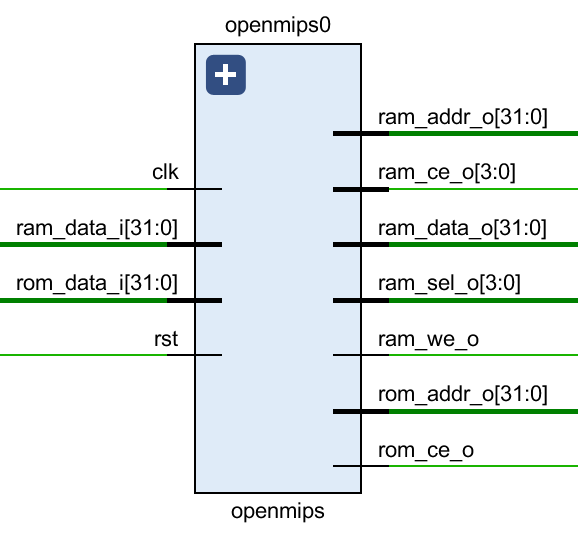


图 Openmips模块

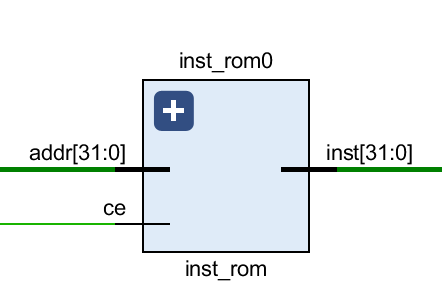


图 inst\_rom模块

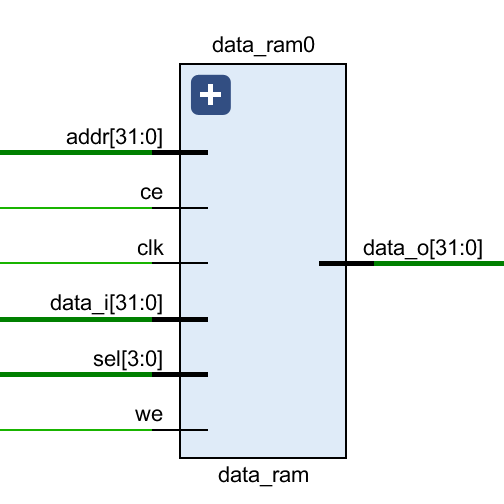


图 data\_ram模块

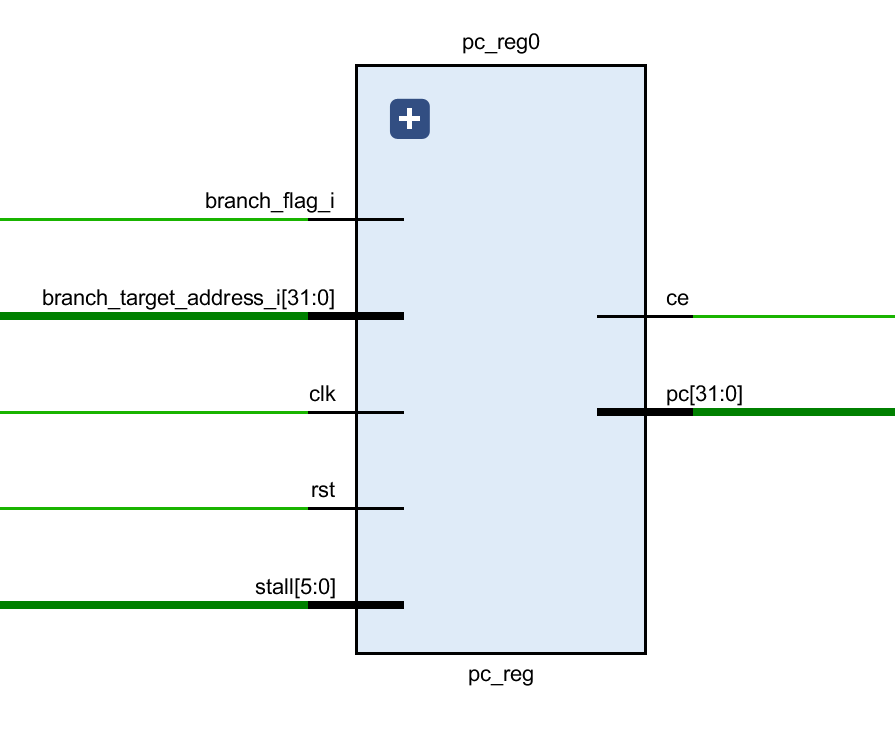


图 pc\_reg模块

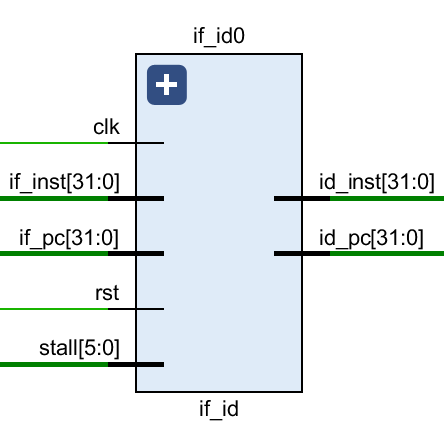


图 if\_id模块

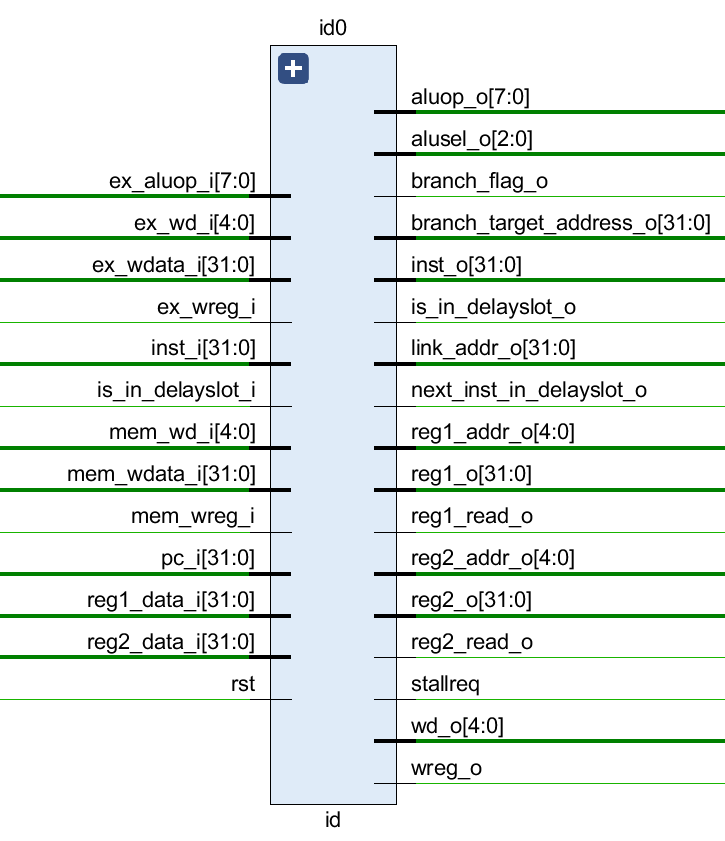


图 id模块

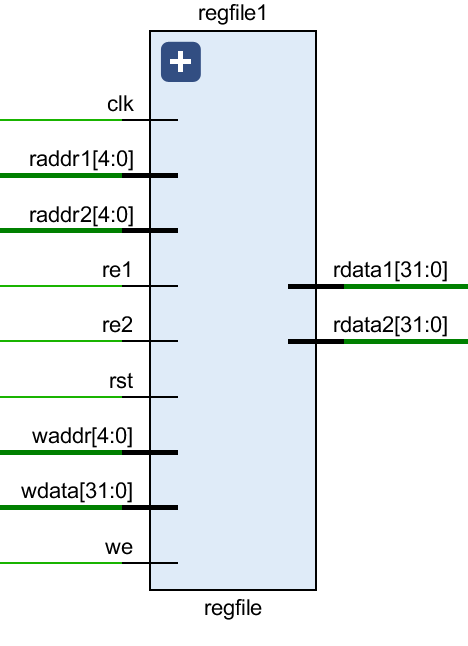


图 regfile模块

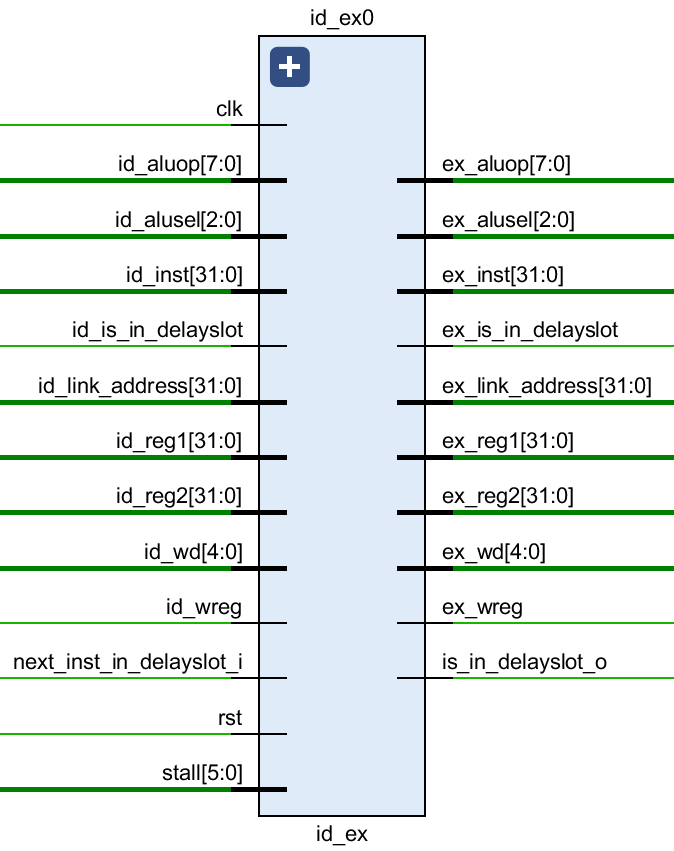


图 id\_ex模块

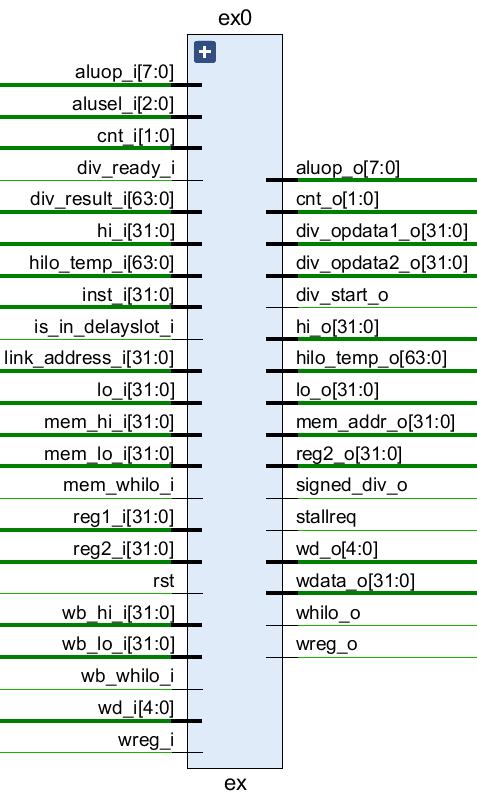


图 ex模块

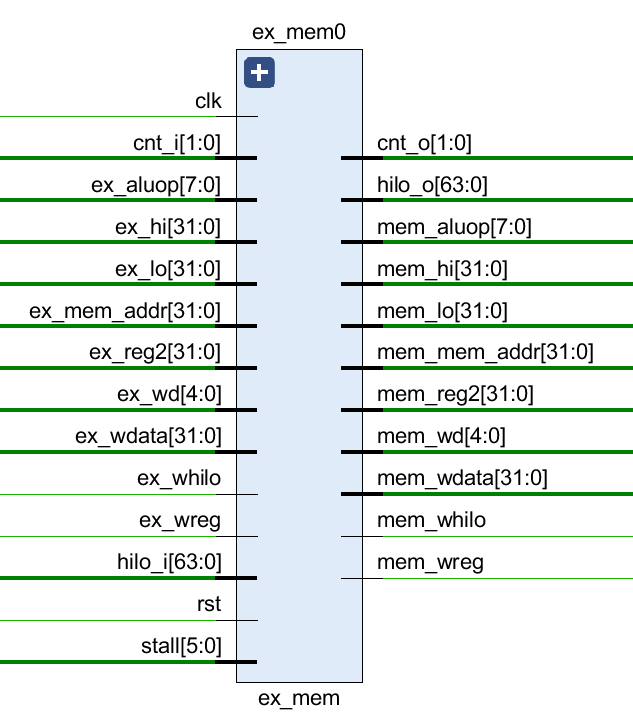


图 ex\_mem模块

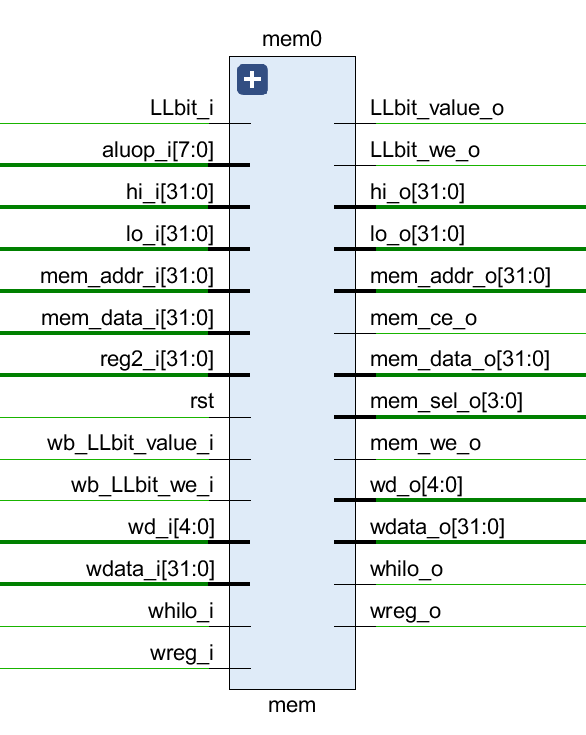


图 mem模块

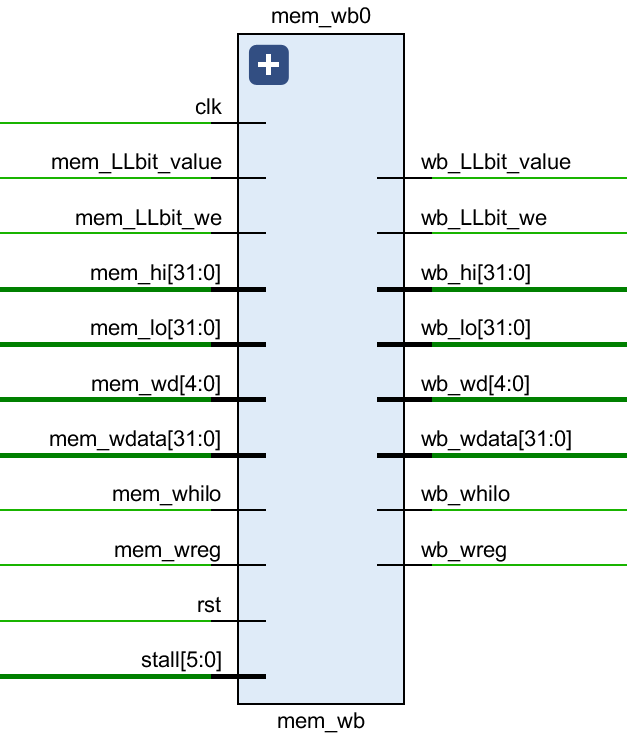


图 mem\_wb模块

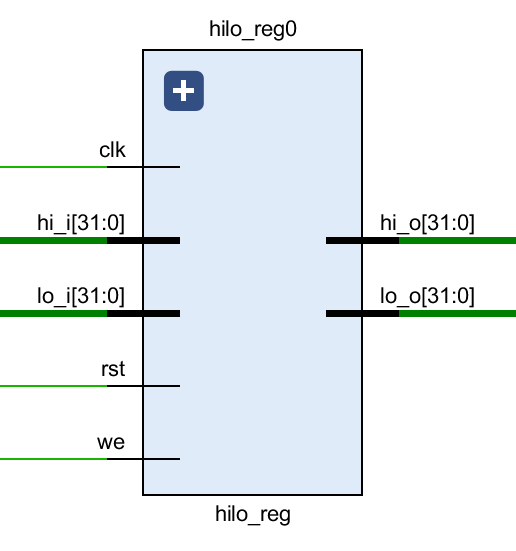


图 hilo\_reg模块

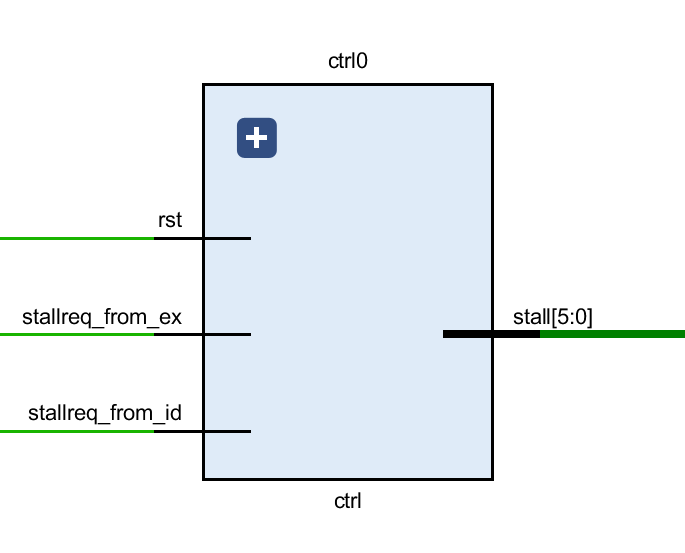


图 ctrl模块

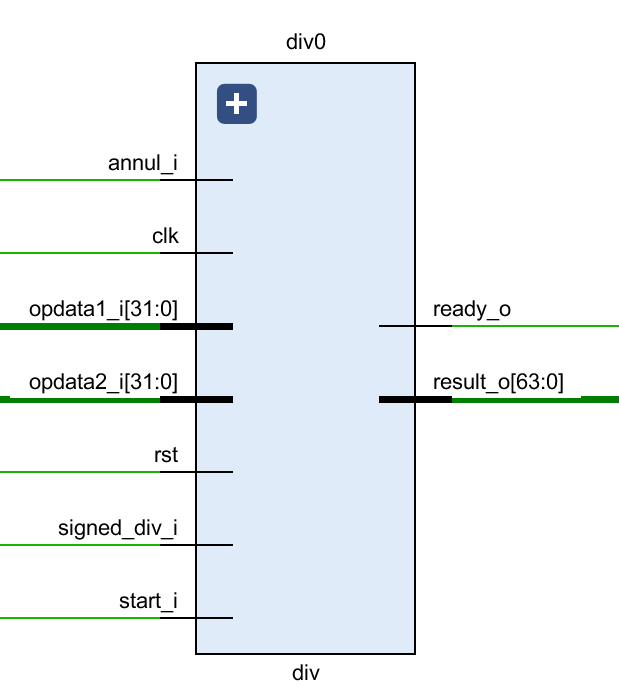


图 div模块

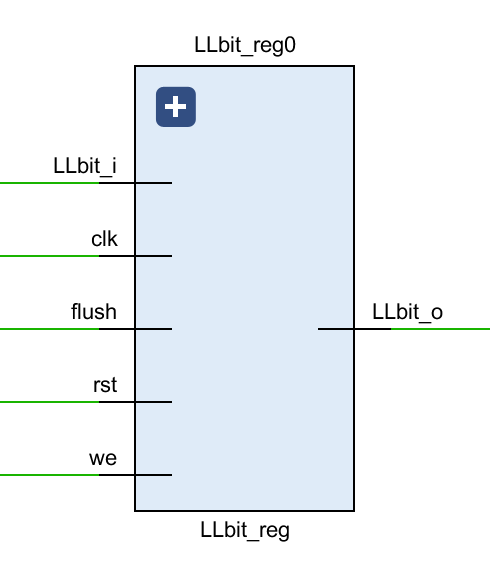


图 LLbit\_reg模块

### 2.4 流水线模块接口描述（包括关键代码）

#### 2.4.3 inst\_rom模块

指令存储器inst\_rom负责存储所有指令，每条指令32位。每当pc自增一次时，便取出4个字节的指令。

表3.1.14 指令存储器 inst\_rom接口描述表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **接口名** | **宽度（bit）** | **输入/输出** | **作用** |
| 1 | ce | 1 | 输入 | 使能信号 |
| 2 | addr | 32 | 输入 | 要读取的指令地址 |
| 3 | inst | 32 | 输出 | 读出的指令 |

#### 2.4.4 data\_ram模块

数据寄存器imma\_ram负责存储数据。共4块131071 x 8位的存储芯片。

表3.1.15 数据存储器 imma\_ram接口描述表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **接口名** | **宽度（bit）** | **输入/输出** | **作用** |
| 1 | ce | 1 | 输入 | 使能信号 |
| 2 | clk | 1 | 输入 | 时钟信号 |
| 3 | imma\_i | 32 | 输入 | 要写入的数据 |
| 4 | addr | 32 | 输入 | 要读取的地址 |
| 5 | we | 1 | 输入 | 是否是写操作 |
| 6 | sel | 4 | 输入 | 字节选择信号 |
| 7 | imma\_o | 32 | 输出 | 读出的数据 |

#### 2.4.5 pc\_reg模块

计数器pc用于取指阶段指示指令地址并顺序递增，一个时钟周期自增32位即4字节。

表3.1.1 计数器pc接口描述表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **接口名** | **宽度（bit）** | **输入/输出** | **作用** |
| 1 | rst | 1 | 输入 | 复位信号 |
| 2 | clk | 1 | 输入 | 时钟信号 |
| 3 | pc | 32 | 输出 | 要读取的指令地址 |
| 4 | ce | 1 | 输出 | 指令存储器使能信号 |
| 5 | branch\_flag\_i | 1 | 输入 | 是否发生转移，如果发生转移，就将pc设置为跳转地址 |
| 6 | branch\_target\_address\_i | 32 | 输入 | 指令转移目标地址 |
| 7 | stall | 6 | 输入 | 是否暂停流水线（由ctrl发出信号决定） |

#### 2.4.6 if\_id模块

if\_id负责暂存取指阶段获得的指令地址和指令，并在下一个时钟周期将它们送入译码阶段。

表3.1.2 if\_id接口描述表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **接口名** | **宽度（bit）** | **输入/输出** | **作用** |
| 1 | rst | 1 | 输入 | 复位信号 |
| 2 | clk | 1 | 输入 | 时钟信号 |
| 3 | if\_inst | 32 | 输入 | 取指阶段的指令 |
| 5 | id\_inst | 32 | 输出 | 指令送入译码阶段 |
| 5 | if\_pc | 32 | 输入 | 取指阶段的地址 |
| 6 | id\_pc | 32 | 输出 | 地址送入译码阶段 |
| 7 | stall | 6 | 输入 | 流水线是否暂停 |

#### 2.4.7 id模块

译码器id负责对指令进行译码，确定运算类型以及运算对象。并从寄存器取出对应数据（如果指令要求），送入到执行阶段。数据相关的接口是为了解决寄存器读取数据有不一致的问题。

表3.1.3 译码器id接口描述表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **接口名** | **宽度** | **输入/输出** | **作用** |
| 1 | rst | 1 | 输入 | 复位信号 |
| 2 | inst\_i | 32 | 输入 | 译码阶段的指令 |
| 3 | reg1\_imma\_i | 32 | 输入 | 从regfile读入寄存器数据1 |
| 4 | reg2\_imma\_i | 32 | 输入 | 从regfile读入寄存器数据2 |
| 5 | aluop\_o | 4 | 输出 | 译码阶段运算类型 |
| 6 | reg1\_o | 32 | 输出 | 译码阶段源操作数1 |
| 7 | reg2\_o | 32 | 输出 | 译码阶段源操作数2 |
| 8 | wd\_o | 5 | 输出 | 目的寄存器地址 |
| 9 | wreg\_o | 1 | 输出 | 是否要写入目的寄存器 |
| 10 | reg2\_addr\_o | 5 | 输出 | regfile第二个寄存器地址 |
| 11 | reg2\_read\_o | 1 | 输出 | regfile第二个寄存器读使能信号 |
| 12 | reg1\_addr\_o | 5 | 输出 | regfile第一个寄存器地址 |
| 13 | reg1\_read\_o | 1 | 输出 | regfile第一个寄存器读使能信号 |
| 14 | inst\_o | 32 | 输出 | 指令输出端口，用于alu计算访存地址mem\_addr\_o |
| **控制相关——延迟槽相关接口** | | | | |
| 15 | is\_in\_delayslot\_i | 1 | 输入 | 当前处于译码阶段的指令是否在延迟槽 |
| 16 | next\_inst\_in\_delayslot\_o | 1 | 输出 | 下一条进入译码阶段的指令是否在延迟槽 |
| 17 | branch\_flag\_o | 1 | 输出 | 是否发生转移 |
| 18 | branch\_target\_address\_o | 32 | 输出 | 转移到的目标地址 |
| 19 | link\_addr\_o | 32 | 输出 | 转移指令保存的返回地址 |
| 20 | is\_in\_delayslot\_o | 1 | 输出 | 当前处于译码阶段的指令是否在延迟槽 |
| **数据相关解决接口** | | | | |
| 21 | ex\_waddr\_i | 5 | 输入 | 执行阶段指令写入寄存器地址 |
| 22 | ex\_wreg\_i | 1 | 输入 | 执行阶段指令是否写寄存器 |
| 23 | ex\_wimma\_i | 32 | 输入 | 执行阶段写入寄存器数据 |
| 24 | mem\_wreg\_i | 1 | 输入 | 访存阶段指令是否写寄存器 |
| 25 | mem\_waddr\_i | 5 | 输入 | 访存阶段指令写入寄存器地址 |
| 26 | mem\_wimma\_i | 32 | 输入 | 访存阶段写入寄存器数据 |

#### 2.4.8 regfile模块

通用寄存器regfile有32个寄存器，其中0号寄存器不使用，负责快速读写32位数据。

表3.1.13 通用寄存器 regfile接口描述表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **接口名** | **宽度（bit）** | **输入/输出** | **作用** |
| 1 | rst | 1 | 输入 | 复位信号 |
| 2 | clk | 1 | 输入 | 时钟信号 |
| 3 | re1 | 1 | 输入 | 读使能信号1 |
| 4 | raddr1 | 5 | 输入 | 读取的寄存器地址1 |
| 5 | re2 | 1 | 输入 | 读使能信号2 |
| 6 | raddr2 | 5 | 输入 | 读取的寄存器地址2 |
| 7 | we | 1 | 输入 | 写使能信号 |
| 8 | waddr | 5 | 输入 | 写入的寄存器地址 |
| 9 | wimma | 32 | 输入 | 写入的数据 |
| 10 | rimma1 | 32 | 输出 | 读出的32位数据1 |
| 11 | rimma2 | 32 | 输出 | 读出的32位数据2 |

#### 2.4.9 id\_ex模块

id\_ex负责将暂存译码器取得的相关数据，例如写目的寄存器地址、数据等，并在下一个时钟周期将它们传送到执行阶段。

表3.1.4 id\_ex接口描述表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **接口名** | **宽度（bit）** | **输入/输出** | **作用** |
| 1 | rst | 1 | 输入 | 复位信号 |
| 2 | clk | 1 | 输入 | 时钟信号 |
| 3 | id\_aluop | 6 | 输入 | 运算类型 |
| 4 | id\_reg1 | 32 | 输入 | 写寄存器1数据 |
| 5 | id\_reg2 | 32 | 输入 | 写寄存器2数据 |
| 6 | id\_wd | 5 | 输入 | 写寄存器地址 |
| 7 | id\_wreg | 1 | 输入 | 是否写寄存器 |
| 8 | ex\_aluop | 6 | 输出 | 运算类型 |
| 9 | ex\_reg1 | 32 | 输出 | 写寄存器1数据 |
| 10 | ex\_reg2 | 32 | 输出 | 写寄存器2数据 |
| 11 | ex\_wd | 5 | 输出 | 写寄存器地址 |
| 12 | ex\_wreg | 1 | 输出 | 是否写寄存器 |
| 13 | id\_inst | 32 | 输入 | 当前指令 |
| 14 | ex\_inst | 32 | 输出 | 当前指令 |
| **分支跳转——延迟槽相关接口** | | | | |
| 15 | id\_link\_address | 32 | 输入 | 返回地址 |
| 16 | id\_is\_in\_delayslot | 1 | 输入 | 当前指令是否在延迟槽 |
| 17 | next\_inst\_in\_delayslot\_i | 1 | 输入 | 下条指令是否在延迟槽 |
| 18 | ex\_link\_address | 32 | 输出 | 返回地址 |
| 19 | ex\_is\_in\_delayslot | 1 | 输出 | 当前指令是否在延迟槽 |
| 20 | is\_in\_delayslot\_o | 1 | 输出 | 下条指令是否在延迟槽 |

#### 2.4.10 ex模块

运算器负责对源操作数进行运算类型要求的运算，例如访存指令计算访存地址，加法指令计算两个数相加，乘法指令则送入乘法器运算等。运算完毕后将运算结果和将写入对象以及地址送入到访存阶段。

表3.1.5 运算器alu接口描述表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **接口名** | **宽度（bit）** | **输入/输出** | **作用** |
| 1 | rst | 1 | 输入 | 复位信号 |
| 2 | alu\_control | 12 | 输入 | 运算类型 |
| 3 | alu\_src1 | 32 | 输入 | 操作数1 |
| 4 | alu\_src2 | 32 | 输入 | 操作数2 |
| 5 | alu\_result | 32 | 输出 | 运算结果 |
| 6 | wd\_i | 5 | 输入 | 写入寄存器地址 |
| 7 | wreg\_i | 1 | 输入 | 是否写入寄存器 |
| 8 | wd\_o | 5 | 输出 | 写入寄存器地址 |
| 9 | wreg\_o | 1 | 输出 | 是否写入寄存器 |
| 10 | inst\_i | 32 | 输入 | 当前指令 |
| 11 | aluop\_o | 6 | 输出 | 运算类型 |
| 12 | mem\_addr\_o | 32 | 输出 | 访存地址 |
| 13 | reg2\_o | 32 | 输出 | 存入存储器的数 |
| 14 | stallreq | 1 | 输出 | 流水线暂停请求->ctrl |
| **延迟槽相关接口** | | | | |
| 15 | link\_address\_i |  | 输入 | 转移指令保存的返回地址 |
| 16 | is\_in\_delayslot\_i | 1 | 输入 | 当前指令是否在延迟槽 |
| **乘法相关接口** | | | | |
| 17 | mul\_result\_i | 64 | 输入 | 乘法运算结果 |
| 18 | mul\_ready\_i | 1 | 输入 | 乘法器是否运算完毕 |
| 19 | mul\_opimma1\_o | 32 | 输出 | 被乘数 |
| 20 | mul\_opimma2\_o | 32 | 输出 | 乘数 |
| 21 | mul\_start\_o | 1 | 输出 | 启动乘法器 |
| 22 | signed\_mul\_o | 1 | 输出 | 有无符号 |
| **除法相关接口** | | | | |
| 23 | div\_result\_i | 64 | 输入 | 除法运算结果 |
| 24 | div\_ready\_i | 1 | 输入 | 除法器是否运算完毕 |
| 25 | div\_opimma1\_o | 32 | 输出 | 被除数 |
| 26 | div\_opimma2\_o | 32 | 输出 | 除数 |
| 27 | div\_start\_o | 1 | 输出 | 启动除法器 |
| 28 | signed\_div\_o | 1 | 输出 | 有无符号 |
| **解决移动指令数据相关的接口** | | | | |
| 29 | hi\_i | 32 | 输入 | HI寄存器内容 |
| 30 | lo\_i | 32 | 输入 | LO寄存器内容 |
| 31 | wb\_hi\_i | 32 | 输入 | 回写时是否要写HI |
| 32 | wb\_lo\_i | 32 | 输入 | 回写时是否要写LO |
| 33 | wb\_whilo\_i | 1 | 输入 | 回写时是否写HI/LO |
| 34 | mem\_hi\_i | 32 | 输入 | 访存时是否要写HI |
| 35 | mem\_lo\_i | 32 | 输入 | 访存时是否要写LO |
| 36 | mem\_whilo\_i | 1 | 输入 | 访存时是否写HI/LO |
| 37 | hi\_o | 32 | 输出 | 执行阶段写入HI的值 |
| 38 | lo\_o | 32 | 输出 | 执行阶段写入LO的值 |
| 39 | whilo\_o | 1 | 输出 | 执行阶段是否要写HI/LO |

#### 2.4.11 ex\_mem模块

ex\_mem负责暂存运算器alu送来的将写对象地址以及数据，并在下一个时钟周期上升沿将这些数据传送到访存阶段。

表3.1.6 计数器pc接口描述表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **接口名** | **宽度（bit）** | **输入/输出** | **作用** |
| 1 | rst | 1 | 输入 | 复位信号 |
| 2 | clk | 1 | 输入 | 时钟信号 |
| 3 | ex\_wd | 5 | 输入 | 写入寄存器地址 |
| 4 | ex\_wreg | 1 | 输入 | 是否写入寄存器 |
| 5 | ex\_wimma | 32 | 输入 | 写入寄存器数据 |
| 6 | mem\_wd | 5 | 输出 | 写入寄存器地址 |
| 7 | mem\_wreg | 1 | 输出 | 是否写入寄存器 |
| 8 | mem\_wimma | 32 | 输出 | 写入寄存器数据 |
| 9 | stall | 6 | 输入 | 流水线暂停 |
| **load/store相关接口** | | | | |
| 10 | ex\_aluop | 6 | 输入 | 运算类型 |
| 11 | ex\_mem\_addr | 32 | 输入 | 访存地址 |
| 12 | ex\_reg2 | 32 | 输入 | 放入存储器的数据 |
| 13 | mem\_aluop | 6 | 输出 | 运算类型 |
| 14 | mem\_mem\_addr | 32 | 输出 | 访存地址 |
| 15 | mem\_reg2 | 32 | 输出 | 放入存储器的数据 |
| **移动指令相关接口** | | | | |
| 16 | ex\_hi | 32 | 输入 | 执行阶段写入HI的值 |
| 17 | ex\_lo | 32 | 输入 | 执行阶段写入LO的值 |
| 18 | ex\_whilo | 1 | 输入 | 执行阶段是否要写HI/LO |
| 19 | mem\_hi | 32 | 输出 | 执行阶段写入HI的值 |
| 20 | mem\_lo | 32 | 输出 | 执行阶段写入LO的值 |
| 21 | mem\_whilo | 1 | 输出 | 执行阶段是否要写HI/LO |

#### 2.4.12 mem模块

访存模块负责和数据存储器RAM的交互，若存在访存相关指令，则将访存相关数据、地址、读写信号等传送至RAM，并获取RAM的相关数据结果等。此外，乘除法运算结果相关数据或者是对寄存器对象的操作相关数据则继续流入下一阶段。

表3.1.7 访存模块mem接口描述表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **接口名** | **宽度（bit）** | **输入/输出** | **作用** |
| 1 | rst | 1 | 输入 | 复位信号 |
| 2 | wd\_i | 12 | 输入 | 写入寄存器地址 |
| 3 | wreg\_i | 1 | 输入 | 是否写入寄存器 |
| 4 | wimma\_i | 32 | 输入 | 写入寄存器数据 |
| 5 | wd\_o | 32 | 输出 | 写入寄存器地址 |
| 6 | wreg\_o | 1 | 输出 | 是否写入寄存器 |
| 7 | wimma\_o | 32 | 输出 | 写入寄存器数据 |
| **load/store相关接口** | | | | |
| 8 | aluop\_i | 6 | 输入 | 运算类型 |
| 9 | mem\_addr\_i | 32 | 输入 | 访存地址 |
| 10 | reg2\_i | 32 | 输入 | 送入存储器的数 |
| **RAM相关接口** | | | | |
| 11 | mem\_imma\_i | 32 | 输入 | 来自RAM的读取数据 |
| 12 | mem\_addr\_o | 32 | 输出 | 写入RAM的地址 |
| 13 | mem\_we\_o | 1 | 输出 | RAM读还是写 1写0读 |
| 14 | mem\_sel\_o | 4 | 输出 | 字节选择信号 |
| 15 | mem\_imma\_o | 32 | 输出 | 存入RAM数据 |
| 16 | mem\_ce\_o | 1 | 输出 | 使能信号 |
| **HI/LO相关接口** | | | | |
| 17 | hi\_i | 32 | 输入 | HI寄存器内容 |
| 18 | lo\_i | 32 | 输入 | LO寄存器内容 |
| 19 | hi\_o | 32 | 输出 | 执行阶段写入HI的值 |
| 20 | lo\_o | 32 | 输出 | 执行阶段写入LO的值 |
| 21 | whilo\_o | 1 | 输出 | 执行阶段是否要写HI/LO |

#### 2.4.13 mem\_wb模块

mem\_wb负责暂存访存阶段暂存对寄存器对象操作的相关数据和乘除法运算结果的相关数据，并在下一个时钟周期上升沿将数据传送至寄存器堆或乘除法专用寄存器。

表3.1.8 mem\_wb接口描述表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **接口名** | **宽度（bit）** | **输入/输出** | **作用** |
| 1 | rst | 1 | 输入 | 复位信号 |
| 2 | clk | 1 | 输入 | 时钟信号 |
| 3 | stall | 6 | 输入 | 流水线暂停 |
| 4 | mem\_hi | 32 | 输入 | 访存时是否要写HI |
| 5 | mem\_lo | 32 | 输入 | 访存时是否要写LO |
| 6 | mem\_whilo | 1 | 输入 | 访存时是否写HI/LO |
| 7 | wb\_hi | 32 | 输出 | 回写时是否要写HI |
| 8 | wb\_lo | 32 | 输出 | 回写时是否要写LO |
| 9 | wb\_whilo | 1 | 输出 | 回写时是否写HI/LO |
| **数据相关解决方案相关接口** | | | | |
| 10 | mem\_wd | 5 | 输入 | 写入寄存器地址 |
| 11 | mem\_wreg | 1 | 输入 | 是否写入寄存器 |
| 12 | mem\_wdata | 32 | 输入 | 写入寄存器数据 |
| 13 | wb\_wd | 5 | 输出 | 写入寄存器地址 |
| 14 | wb\_wreg | 1 | 输出 | 是否写入寄存器 |
| 15 | wb\_wdata | 32 | 输出 | 写入寄存器数据 |

#### 2.4.14 hilo\_reg模块

乘除法专用寄存器hilo\_reg负责存储乘除法运算结果的两个寄存器，对于除法来说，HI存余数，LO存商，对于乘法来说，HI存高32位运算结果，LO存低32位运算结果。

表3.1.12 乘除法专用寄存器 hilo\_reg接口描述表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **接口名** | **宽度（bit）** | **输入/输出** | **作用** |
| 1 | rst | 1 | 输入 | 复位信号 |
| 2 | clk | 1 | 输入 | 时钟信号 |
| 3 | we | 1 | 输入 | 写使能信号 |
| 4 | hi\_i | 32 | 输入 | 输入的HI值 |
| 5 | lo\_i | 32 | 输入 | 输入的LO值 |
| 6 | hi\_o | 32 | 输入 | 输出的HI值 |
| 7 | lo\_o | 32 | 输出 | 输出的LO值 |

#### 2.4.15 ctrl模块

流水线控制器 ctrl 负责收集从译码阶段或访存阶段发来的流水线暂停和恢复请求，并执行对应的流水线暂停和恢复操作。

表3.1.9 流水线控制器 ctrl接口描述表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **接口名** | **宽度（bit）** | **输入/输出** | **作用** |
| 1 | rst | 1 | 输入 | 复位信号 |
| 2 | stallreq\_from\_id | 1 | 输入 | 译码阶段流水线暂停请求 |
| 3 | stallreq\_from\_ex | 1 | 输入 | 执行阶段流水线暂停请求 |
| 4 | stall | 6 | 输出 | 流水线暂停信号 |

#### 2.4.16 div模块

除法器使用试商法进行除法运算，期间会暂停流水线，运算完毕后将结果输出至运算器alu。

表3.1.10 除法器 div接口描述表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **接口名** | **宽度（bit）** | **输入/输出** | **作用** |
| 1 | rst | 1 | 输入 | 复位信号 |
| 2 | clk | 1 | 输入 | 时钟信号 |
| 3 | signed\_div\_i | 1 | 输入 | 有无符号 |
| 4 | opimma1\_i | 32 | 输入 | 被除数 |
| 5 | opimma2\_i | 32 | 输入 | 除数 |
| 6 | start\_i | 1 | 输入 | 是否开始除法运算 |
| 7 | annul\_i | 1 | 输入 | 是否取消除法运算 1是 |
| 8 | result\_o | 64 | 输出 | 运算结果 |
| 9 | ready\_o | 1 | 输出 | 除法运算结束标志 |

#### 2.4.17 LLbit\_reg模块

表3.1.10 除法器 div接口描述表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **接口名** | **宽度（bit）** | **输入/输出** | **作用** |
| 1 | rst | 1 | 输入 | 复位信号 |
| 2 | clk | 1 | 输入 | 时钟信号 |
| 3 | signed\_div\_i | 1 | 输入 | 有无符号 |
| 4 | opimma1\_i | 32 | 输入 | 被除数 |
| 5 | opimma2\_i | 32 | 输入 | 除数 |
| 6 | start\_i | 1 | 输入 | 是否开始除法运算 |
| 7 | annul\_i | 1 | 输入 | 是否取消除法运算 1是 |
| 8 | result\_o | 64 | 输出 | 运算结果 |
| 9 | ready\_o | 1 | 输出 | 除法运算结束标志 |

#### 2.4.18 各模块关键代码

|  |
| --- |
| openmips\_min\_sopc模块 |
| module openmips\_min\_sopc(  input wire clk,rst  );  wire[`InstAddrBus] inst\_addr;  wire[`InstBus] inst;  wire rom\_ce;  wire mem\_we\_i;  wire[`RegBus] mem\_addr\_i;  wire[`RegBus] mem\_data\_i;  wire[`RegBus] mem\_data\_o;  wire[3:0] mem\_sel\_i;  wire mem\_ce\_i;  openmips openmips0(  .clk(clk),  .rst(rst),  .rom\_addr\_o(inst\_addr),  .rom\_data\_i(inst),  .rom\_ce\_o(rom\_ce),  .ram\_we\_o(mem\_we\_i),  .ram\_addr\_o(mem\_addr\_i),  .ram\_sel\_o(mem\_sel\_i),  .ram\_data\_o(mem\_data\_i),  .ram\_data\_i(mem\_data\_o),  .ram\_ce\_o(mem\_ce\_i)  );    inst\_rom inst\_rom0(  .ce(rom\_ce),  .addr(inst\_addr),  .inst(inst)  );  data\_ram data\_ram0(  .clk(clk),  .ce(mem\_ce\_i),  .we(mem\_we\_i),  .addr(mem\_addr\_i),  .sel(mem\_sel\_i),  .data\_i(mem\_data\_i),  .data\_o(mem\_data\_o)  ); |

|  |
| --- |
| openmips模块 |
| module openmips(  input wire clk,rst,  input wire[`RegBus] rom\_data\_i,  output wire[`RegBus]rom\_addr\_o,  output wire rom\_ce\_o,    input wire[`RegBus] ram\_data\_i,  output wire[`RegBus] ram\_addr\_o,  output wire[`RegBus] ram\_data\_o,  output wire ram\_we\_o,  output wire[3:0] ram\_sel\_o,  output wire[3:0] ram\_ce\_o  ); |

|  |
| --- |
| inst\_rom模块 |
| module inst\_rom(  input wire ce,  input wire[`InstAddrBus]addr,  output reg[`InstBus]inst  ); |

|  |
| --- |
| data\_ram模块 |
| module data\_ram(  input wire clk,ce,we,  input wire[`DataAddrBus] addr,  input wire[3:0] sel,  input wire[`DataBus] data\_i,  output reg[`DataBus] data\_o  ); |

|  |
| --- |
| pc\_reg模块 |
| module pc\_reg(  input wire clk,rst,  input wire[5:0] stall,  input wire branch\_flag\_i,  input wire[`RegBus] branch\_target\_address\_i,    output reg[`InstAddrBus] pc,  output reg ce  ); |

|  |
| --- |
| if\_id模块 |
| module if\_id(  input wire clk,rst,  input wire[5:0] stall,  input wire[`InstAddrBus] if\_pc,  input wire[`InstBus] if\_inst,  output reg[`InstAddrBus] id\_pc,  output reg[`InstBus] id\_inst    ); |

|  |
| --- |
| id模块 |
| module id(  input wire rst,  input wire[`InstAddrBus] pc\_i,  input wire[`InstBus] inst\_i,  input wire[`AluOpBus] ex\_aluop\_i,  input wire ex\_wreg\_i,  input wire[`RegBus] ex\_wdata\_i,  input wire[`RegAddrBus] ex\_wd\_i,    input wire mem\_wreg\_i,  input wire[`RegBus] mem\_wdata\_i,  input wire[`RegAddrBus] mem\_wd\_i,    input wire[`RegBus] reg1\_data\_i,  input wire[`RegBus] reg2\_data\_i,  input wire is\_in\_delayslot\_i,  output reg reg1\_read\_o,  output reg reg2\_read\_o,  output reg[`RegAddrBus] reg1\_addr\_o,  output reg[`RegAddrBus] reg2\_addr\_o,    output reg[`AluOpBus] aluop\_o,  output reg[`AluSelBus] alusel\_o,  output reg[`RegBus] reg1\_o,  output reg[`RegBus] reg2\_o,  output reg[`RegAddrBus] wd\_o,  output reg wreg\_o,  output wire[`RegBus] inst\_o,  output reg next\_inst\_in\_delayslot\_o,    output reg branch\_flag\_o,  output reg[`RegBus] branch\_target\_address\_o,  output reg[`RegBus] link\_addr\_o,  output reg is\_in\_delayslot\_o,    output wire stallreq  ); |

|  |
| --- |
| regfile模块 |
| module regfile(  input wire clk,rst,we,  input wire[`RegAddrBus] waddr,  input wire[`RegBus] wdata,    //?????1  input wire re1,  input wire[`RegAddrBus] raddr1,  output reg[`RegBus] rdata1,    //?????2  input wire re2,  input wire[`RegAddrBus] raddr2,  output reg[`RegBus] rdata2    ); |

|  |
| --- |
| id\_ex模块 |
| module id\_ex(  input wire clk,rst,  input wire[5:0] stall,    input wire[`AluOpBus] id\_aluop,  input wire[`AluSelBus] id\_alusel,  input wire[`RegBus] id\_reg1,  input wire[`RegBus] id\_reg2,  input wire[`RegAddrBus] id\_wd,  input wire id\_wreg,  input wire[`RegBus] id\_link\_address,  input wire id\_is\_in\_delayslot,  input wire next\_inst\_in\_delayslot\_i,  input wire[`RegBus] id\_inst,    output reg[`AluOpBus] ex\_aluop,  output reg[`AluSelBus] ex\_alusel,  output reg[`RegBus] ex\_reg1,  output reg[`RegBus] ex\_reg2,  output reg[`RegAddrBus] ex\_wd,  output reg ex\_wreg,  output reg[`RegBus] ex\_link\_address,  output reg ex\_is\_in\_delayslot,  output reg is\_in\_delayslot\_o,  output reg[`RegBus] ex\_inst    ); |

|  |
| --- |
| ex模块 |
| module ex(  input wire rst,    input wire[`AluOpBus] aluop\_i,  input wire[`AluSelBus] alusel\_i,  input wire[`RegBus] reg1\_i,  input wire[`RegBus] reg2\_i,  input wire[`RegAddrBus] wd\_i,  input wire wreg\_i,  input wire[`RegBus] inst\_i,    input wire[`RegBus] hi\_i,  input wire[`RegBus] lo\_i,  input wire[`RegBus] wb\_hi\_i,  input wire[`RegBus] wb\_lo\_i,  input wire wb\_whilo\_i,    input wire[`RegBus] mem\_hi\_i,  input wire[`RegBus] mem\_lo\_i,  input wire mem\_whilo\_i,  input wire[`DoubleRegBus] hilo\_temp\_i,  input wire[1:0] cnt\_i,  input wire[`DoubleRegBus] div\_result\_i,  input wire div\_  input wire[`RegBus] link\_address\_i,  input wire is\_in\_delayslot\_i,    output reg[`RegAddrBus] wd\_o,  output reg wreg\_o,  output reg[`RegBus] wdata\_o,  output reg[`RegBus] hi\_o,  output reg[`RegBus] lo\_o,  output reg whilo\_o,    output reg[`DoubleRegBus] hilo\_temp\_o,  output reg[1:0] cnt\_o,  output reg[`RegBus] div\_opdata1\_o,  output reg[`RegBus] div\_opdata2\_o,  output reg div\_start\_o,  output reg signed\_div\_o,  output wire[`AluOpBus] aluop\_o,  output wire[`RegBus] mem\_addr\_o,  output wire[`RegBus] reg2\_o,  output reg stallreq    ); |

|  |
| --- |
| ex\_mem模块 |
| module ex\_mem(  input wire clk,rst,    input wire[5:0] stall,    input wire[`RegAddrBus] ex\_wd,  input wire ex\_wreg,  input wire[`RegBus] ex\_wdata,  input wire[`RegBus] ex\_hi,  input wire[`RegBus] ex\_lo,  input wire ex\_whilo,  input wire[`AluOpBus] ex\_aluop,  input wire[`RegBus] ex\_mem\_addr,  input wire[`RegBus] ex\_reg2,  input wire[`DoubleRegBus] hilo\_i,  input wire[1:0] cnt\_i,    output reg[`RegAddrBus] mem\_wd,  output reg mem\_wreg,  output reg[`RegBus] mem\_wdata,  output reg[`RegBus] mem\_hi,  output reg[`RegBus] mem\_lo,  output reg mem\_whilo,  output reg[`AluOpBus] mem\_aluop,  output reg[`RegBus] mem\_mem\_addr,  output reg[`RegBus] mem\_reg2,    output reg[`DoubleRegBus] hilo\_o,  output reg[1:0] cnt\_o      ); |

|  |
| --- |
| mem模块 |
| module mem(  input wire rst,    input wire[`RegAddrBus] wd\_i,  input wire wreg\_i,  input wire[`RegBus] wdata\_i,  input wire[`RegBus] hi\_i,  input wire[`RegBus] lo\_i,  input wire whilo\_i,  input wire[`AluOpBus] aluop\_i,  input wire[`RegBus] mem\_addr\_i,  input wire[`RegBus] reg2\_i,    input wire[`RegBus] mem\_data\_i,  input wire LLbit\_i,  input wire wb\_LLbit\_we\_i,  input wire wb\_LLbit\_value\_i,    output reg[`RegAddrBus] wd\_o,  output reg wreg\_o,  output reg[`RegBus] wdata\_o,  output reg[`RegBus] hi\_o,  output reg[`RegBus] lo\_o,  output reg whilo\_o,  output reg LLbit\_we\_o,  output reg LLbit\_value\_o,    output reg[`RegBus] mem\_addr\_o,  output wire mem\_we\_o,  output reg[3:0] mem\_sel\_o,  output reg[`RegBus] mem\_data\_o,  output reg mem\_ce\_o    ); |

|  |
| --- |
| mem\_wb模块 |
| module mem\_wb(  input wire clk,rst,  input wire[5:0] stall,  input wire[`RegAddrBus] mem\_wd,  input wire mem\_wreg,  input wire[`RegBus] mem\_wdata,  input wire[`RegBus] mem\_hi,  input wire[`RegBus] mem\_lo,  input wire mem\_whilo,    input wire mem\_LLbit\_we,  input wire mem\_LLbit\_value,  output reg[`RegAddrBus] wb\_wd,  output reg wb\_wreg,  output reg[`RegBus] wb\_wdata,  output reg[`RegBus] wb\_hi,  output reg[`RegBus] wb\_lo,  output reg wb\_whilo,  output reg wb\_LLbit\_we,  output reg wb\_LLbit\_value    ); |

|  |
| --- |
| hilo\_reg模块 |
| module hilo\_reg(  input wire clk,rst,we,  input wire[`RegBus] hi\_i,  input wire[`RegBus] lo\_i,  output reg[`RegBus] hi\_o,  output reg[`RegBus] lo\_o    ); |

|  |
| --- |
| ctrl模块 |
| module ctrl(  input wire rst,  input wire stallreq\_from\_id,  input wire stallreq\_from\_ex,  output reg[5:0] stall  ); |

|  |
| --- |
| div模块 |
| module div(  input wire clk,rst,    input wire signed\_div\_i,  input wire[31:0] opdata1\_i,  input wire[31:0] opdata2\_i,  input wire start\_i,  input wire annul\_i,    output reg[63:0] result\_o,  output reg ready\_o  ); |

|  |
| --- |
| LLbit\_reg模块 |
| module LLbit\_reg(  input wire clk,rst,flush,LLbit\_i,we,  output reg LLbit\_o  ); |

### 2.5 综合结果

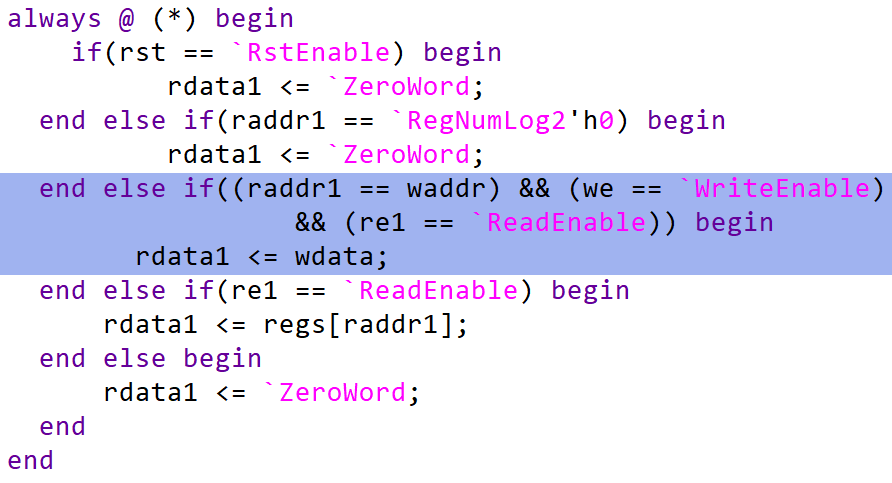
#### 2.5.1 数据相关

流水线数据相关又分为三种情况：RAW、WAR、WAW。

对于五级流水CPU来说，只有在流水线回写阶段才会写寄存器，因此不存在WAW相关。又因为只能在流水线译码阶段读寄存器、回写阶段写寄存器，不存在WAR相关。所以五级流水CPU只存在RAW相关，RAW相关又有三种情况：

1. 相邻指令间存在数据相关。
2. 相隔1条指令的指令间存在数据相关。
3. 相隔2条指令的指令间存在数据相关。

相隔2条指令的RAW相关问题在Regfile模块中已经解决，如下图所示：

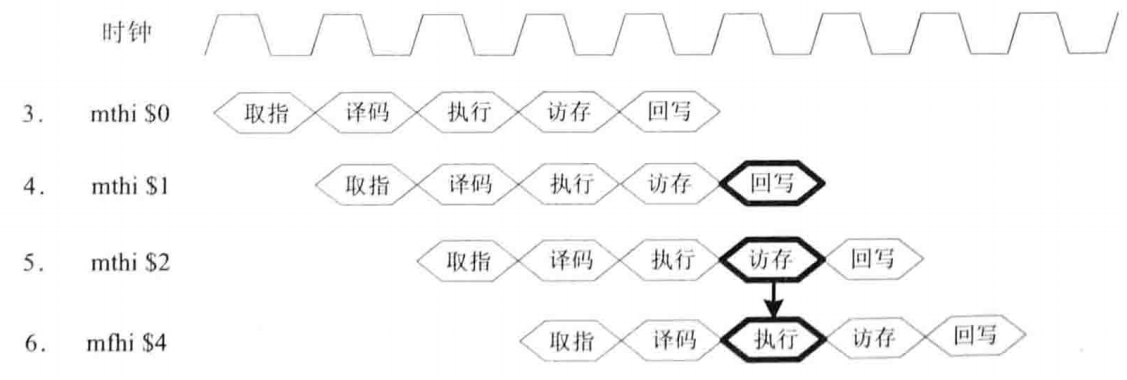


而相邻指令间的RAW和相隔1条指令的RAW需要通过数据前推方式解决。

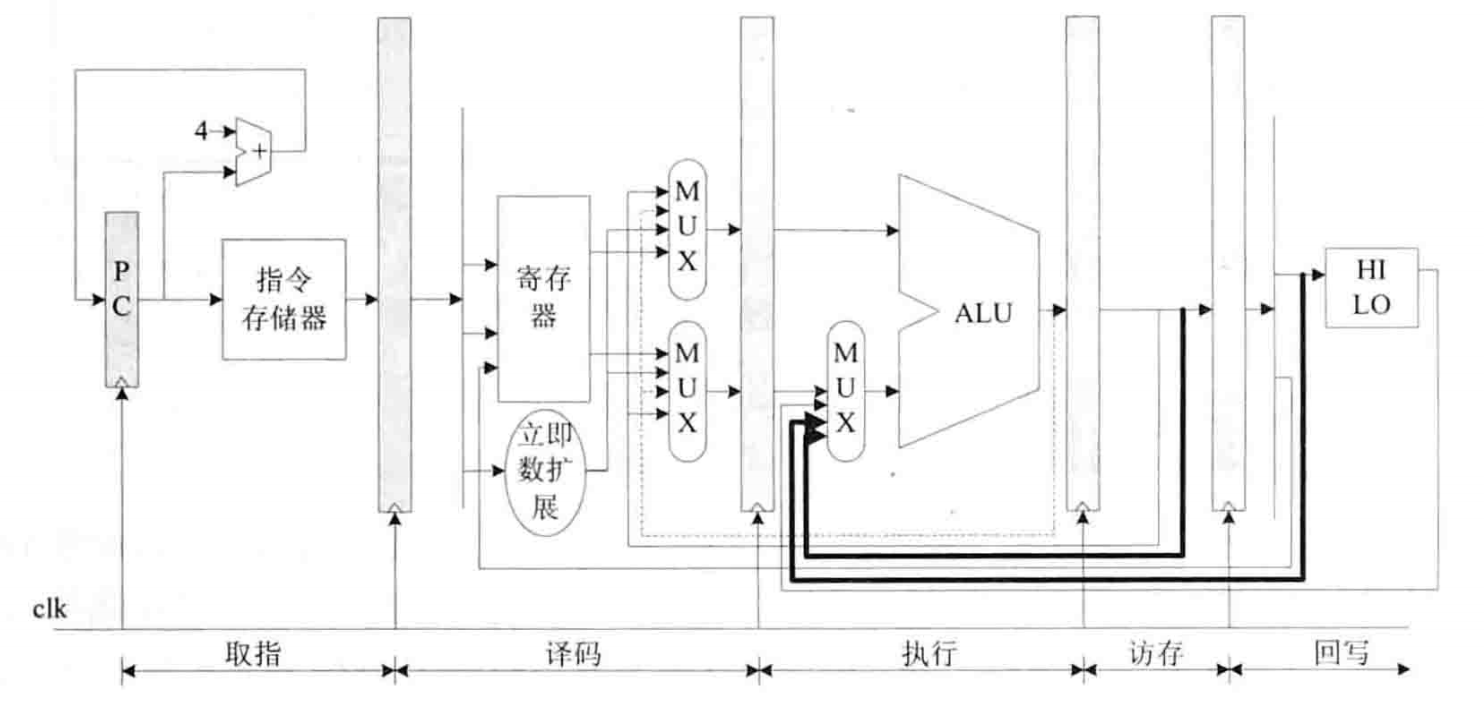
|  |
| --- |
| **Id模块的数据前推实现** |
| module id(  //其他部分省略  input wire ex\_wreg\_i,  input wire[`RegBus] ex\_wdata\_i,  input wire[`RegAddrBus] ex\_wd\_i,  input wire mem\_wreg\_i,  input wire[`RegBus] mem\_wdata\_i,  input wire[`RegAddrBus] mem\_wd\_i,);  always @ (\*) begin  stallreq\_for\_reg2\_loadrelate <= `NoStop;  if(rst == `RstEnable) begin  reg2\_o <= `ZeroWord;  end else if(pre\_inst\_is\_load == 1'b1 && ex\_wd\_i == reg2\_addr\_o  && reg2\_read\_o == 1'b1 ) begin  stallreq\_for\_reg2\_loadrelate <= `Stop;  end else if((reg2\_read\_o == 1'b1) && (ex\_wreg\_i == 1'b1)  && (ex\_wd\_i == reg2\_addr\_o)) begin  reg2\_o <= ex\_wdata\_i;  end else if((reg2\_read\_o == 1'b1) && (mem\_wreg\_i == 1'b1)  && (mem\_wd\_i == reg2\_addr\_o)) begin  reg2\_o <= mem\_wdata\_i;  end else if(reg2\_read\_o == 1'b1) begin  reg2\_o <= reg2\_data\_i;  end else if(reg2\_read\_o == 1'b0) begin  reg2\_o <= imm;  end else begin  reg2\_o <= `ZeroWord;  end  end |

通过在ID模块加入一些输入接口，可以解决数据冲突问题，当发生读后写冲突时，EX模块和MEM模块会将数据前推至ID模块，从而读到正确的写之后的寄存器值。

然而，在后续的添加指令过程中出现了新的数据相关问题，mfhi、mflo指令会造成执行时获得的寄存器值出现冲突，如下图所示：

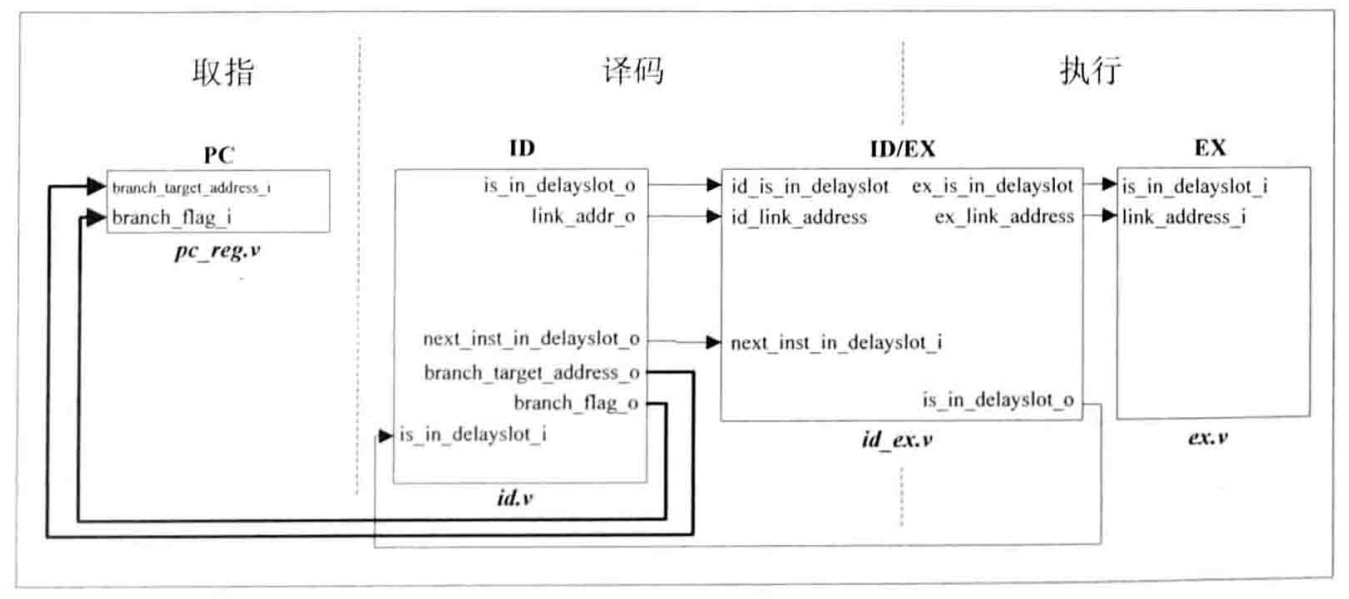


为了解决该RAW问题，我增加了HILO模块，用于实现HI、LO寄存器，并且在EX阶段可以接收该寄存器发来的值，从而解决了HI、LO寄存器的数据相关问题。如下图数据流图所示：

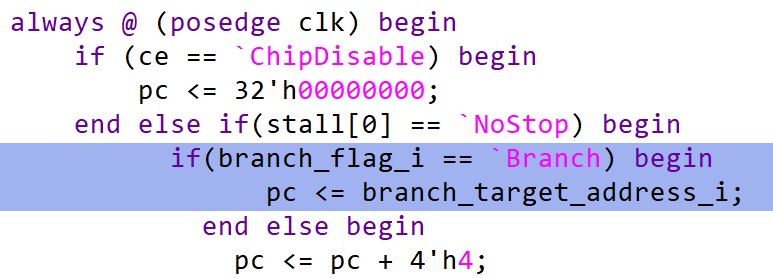


#### 2.5.2 控制相关

控制相关由分支指令造成，通过延迟槽技术可以完美解决。由于分支指令的出现，导致了流水线前面的几个阶段可能会失效，为了解决这个问题，提出了延迟槽技术，并且将分支指令要转向的那个指令以及是否转移的问题都放在了ID译码模块，从而我们可以及时通知PC做出转移操作。具体的实现数据流图如下图所示：



在PC模块，接收分支跳转指令的信号，如果接收到跳转信号，则PC的值就为接收到的要跳转的地址，关键代码如下图所示：



而在ID模块，如果

#### 2.5.3 结构相关

结构相关即某一级流水停顿了，会阻塞上游的流水级。

# 【系统测试】

## 1、 数字跑表系统测试

### 1. 数字跑表的功能仿真

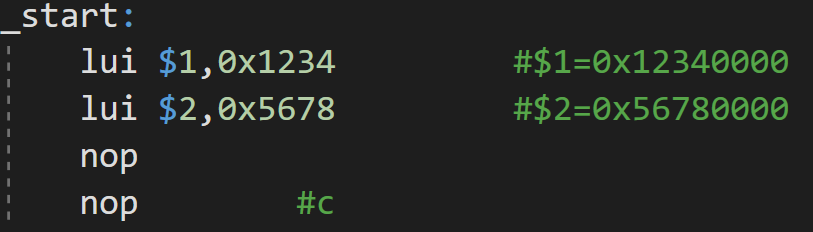
### 2. 数字跑表的RTL级逻辑电路

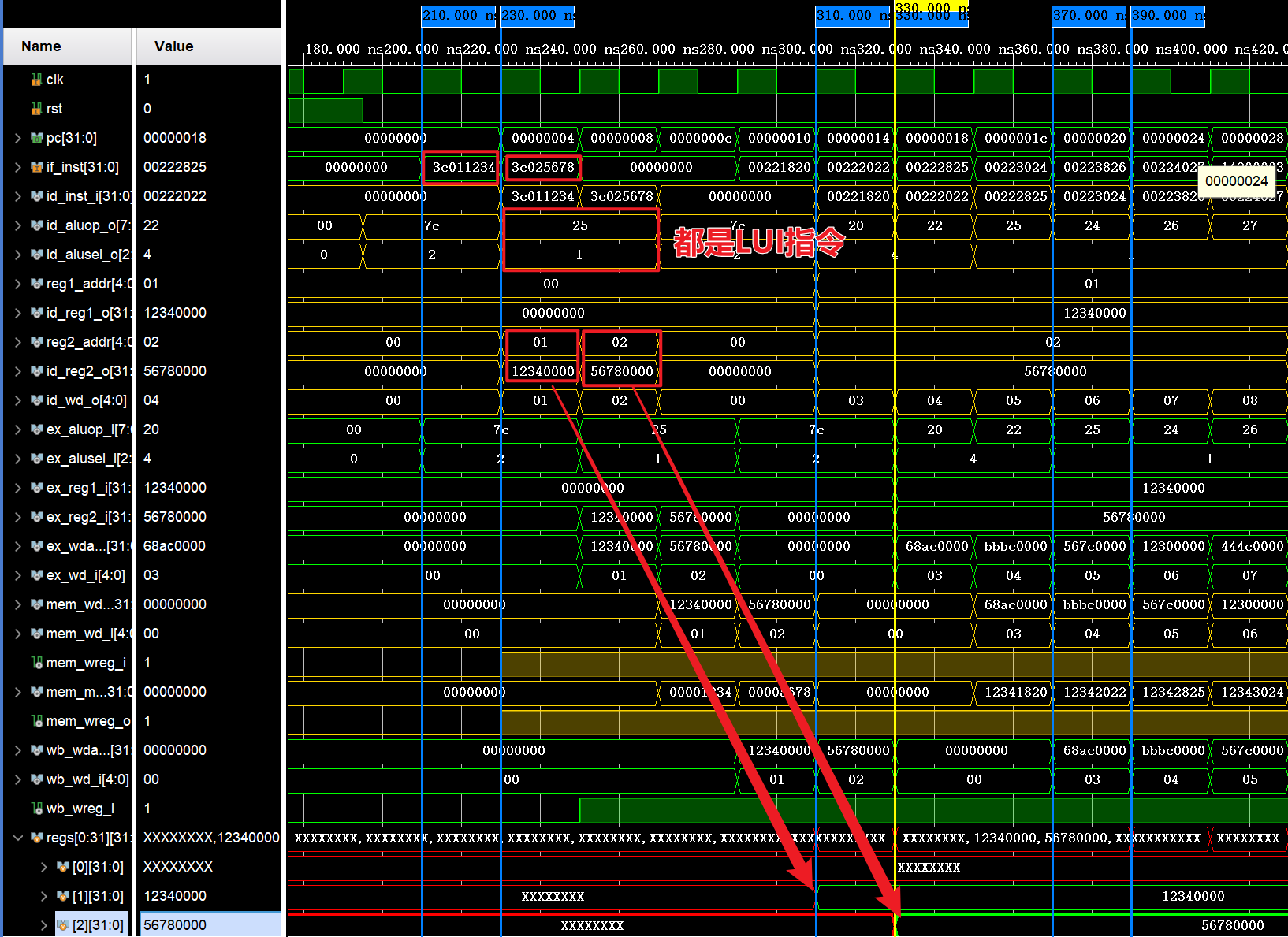
## 五级流水CPU系统测试

### 1. 五级流水CPU的功能仿真

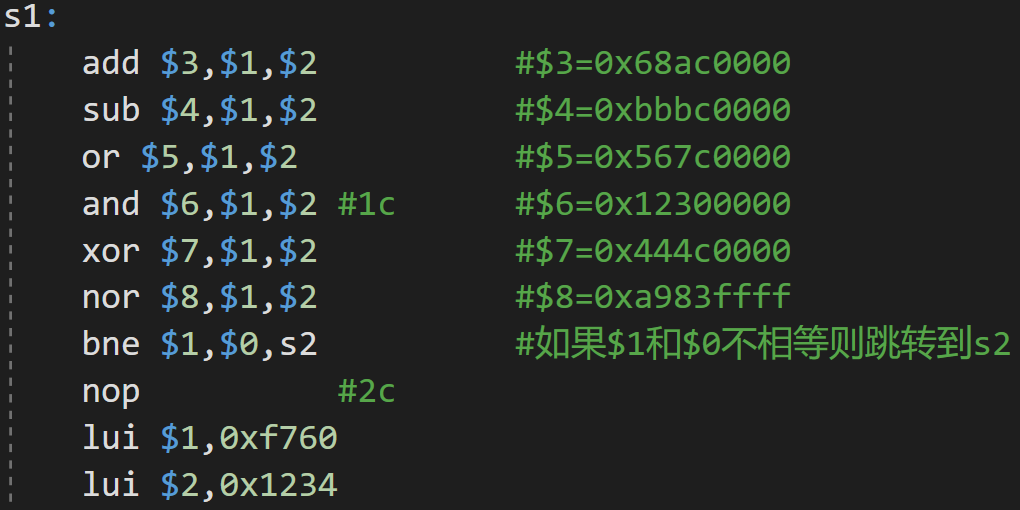
#### 测试用例1（func\_test1）

首先是两条LUI指令，为1号、2号寄存器赋初值，如下图所示

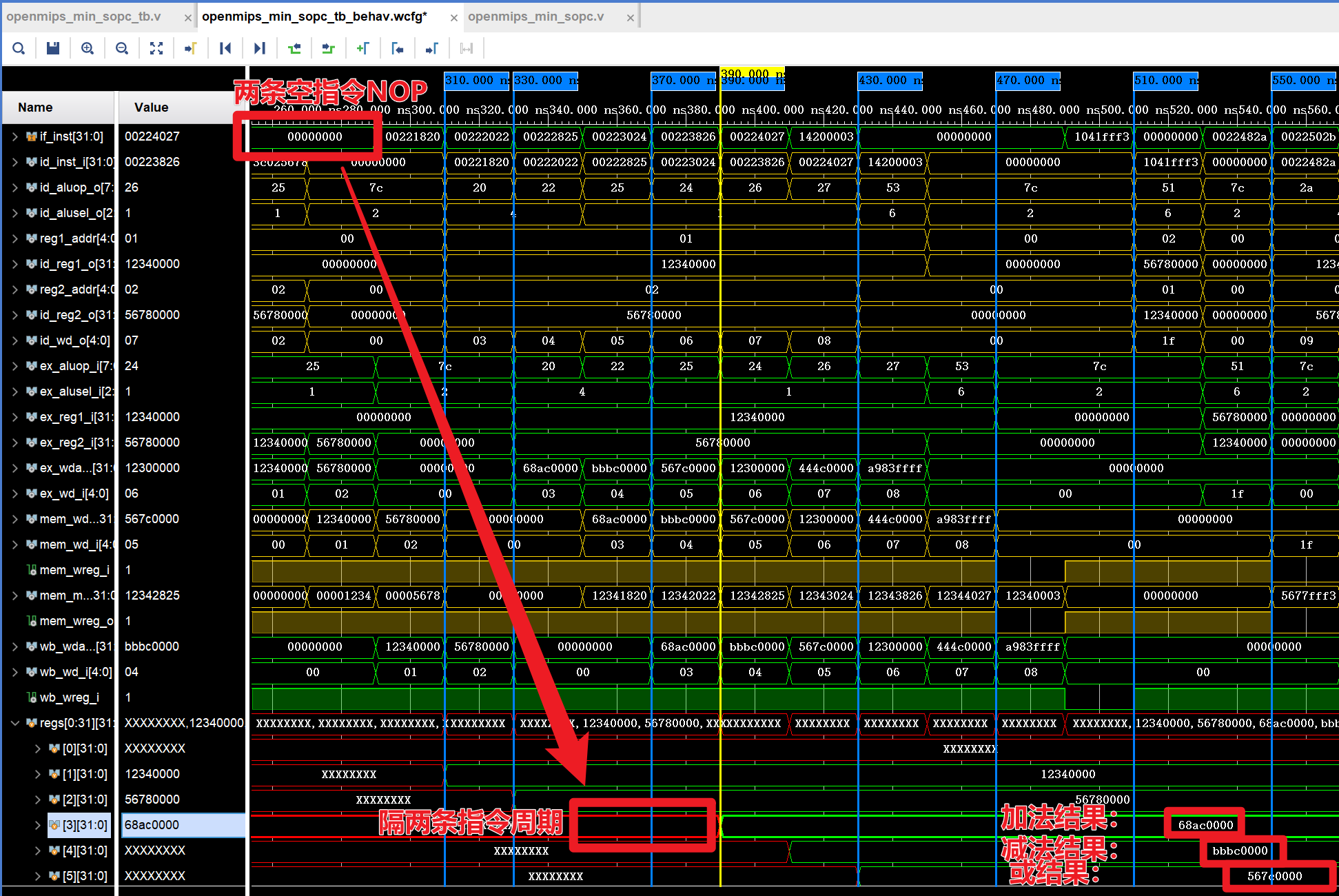


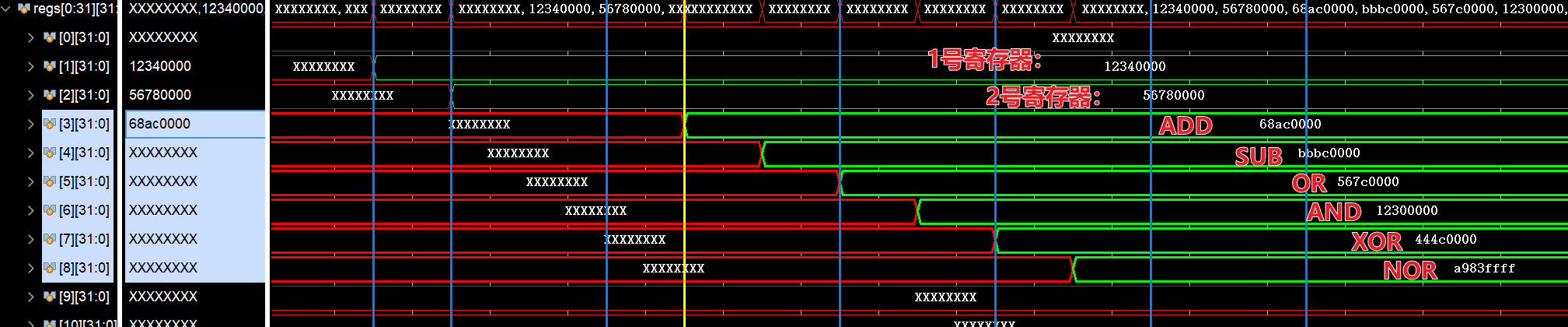


然后执行s1处代码，如下图所示：

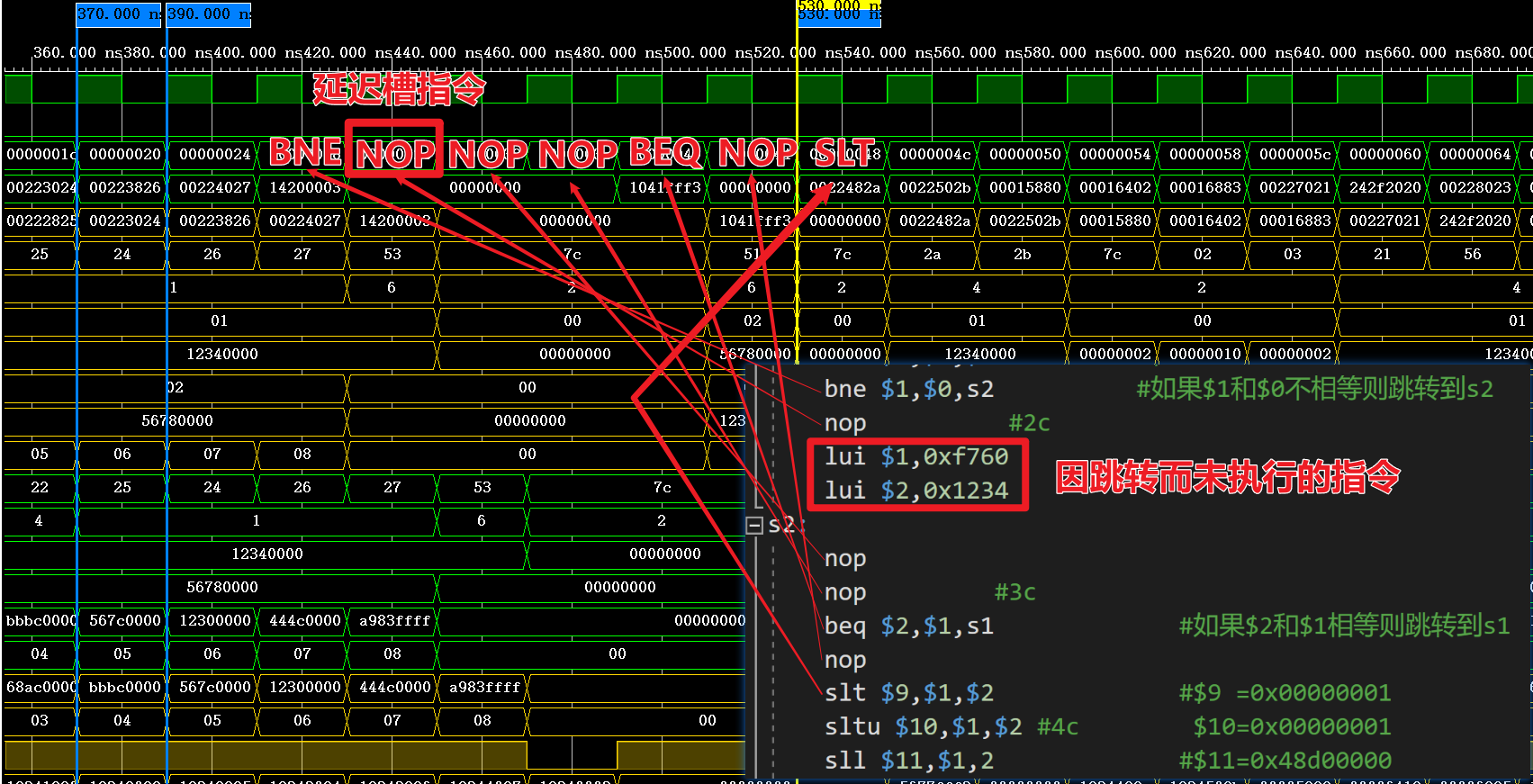


首先加法结果存入3号寄存器，将减法结果存入4号寄存器，OR结果存入5号寄存器，AND结果存入6号寄存器，XOR结果存入7号寄存器，NOR结果存入8号寄存器，具体结果如下图所示：

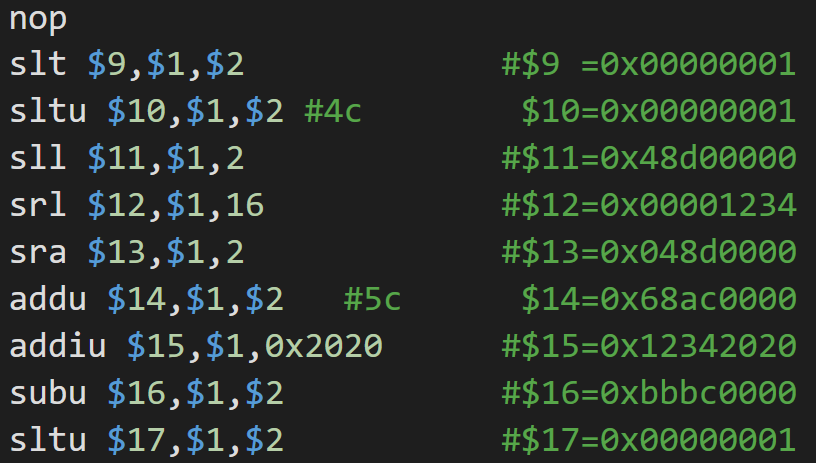




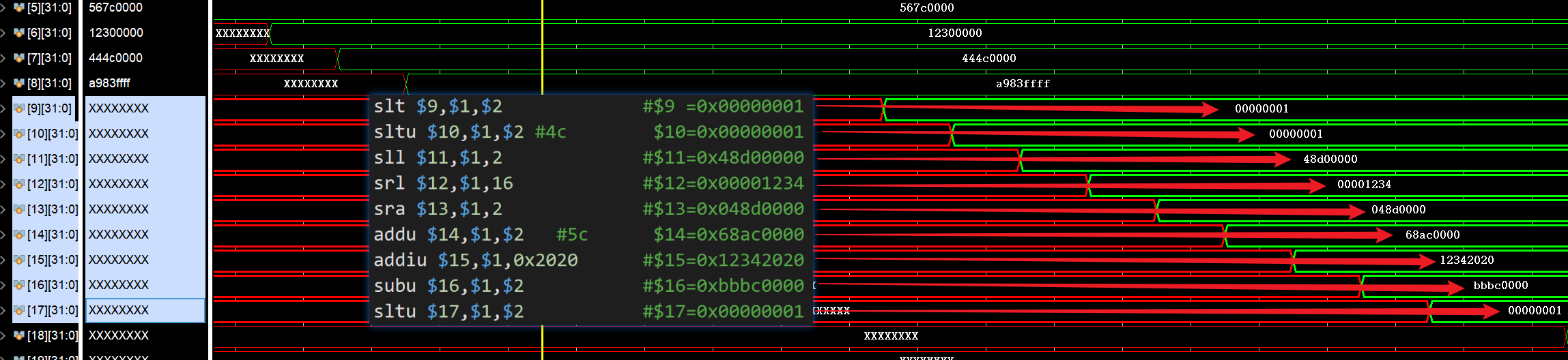
紧接着是一个跳转指令BNE，具体的跳转分析及延迟槽技术分析如下图所示：



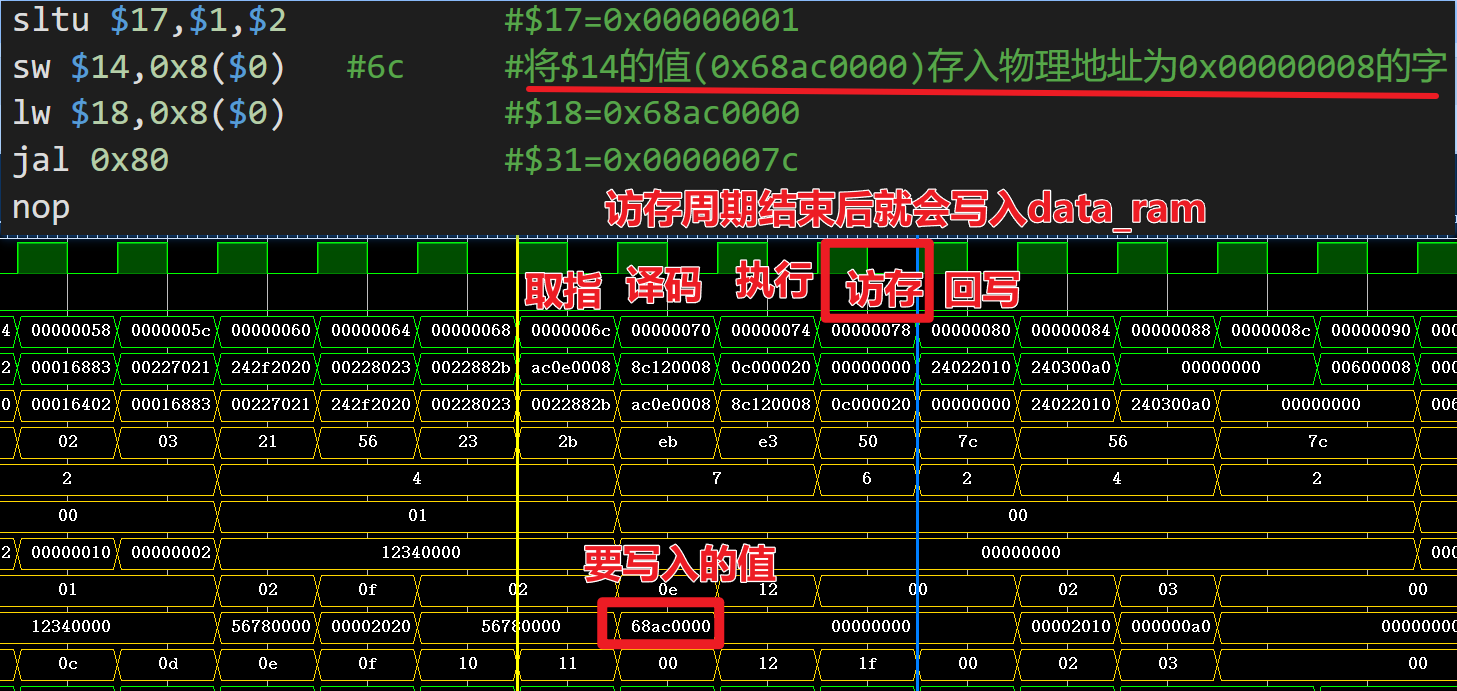
然后s2代码最先是一系列运算，并存入相应的寄存器中，如下图所示：



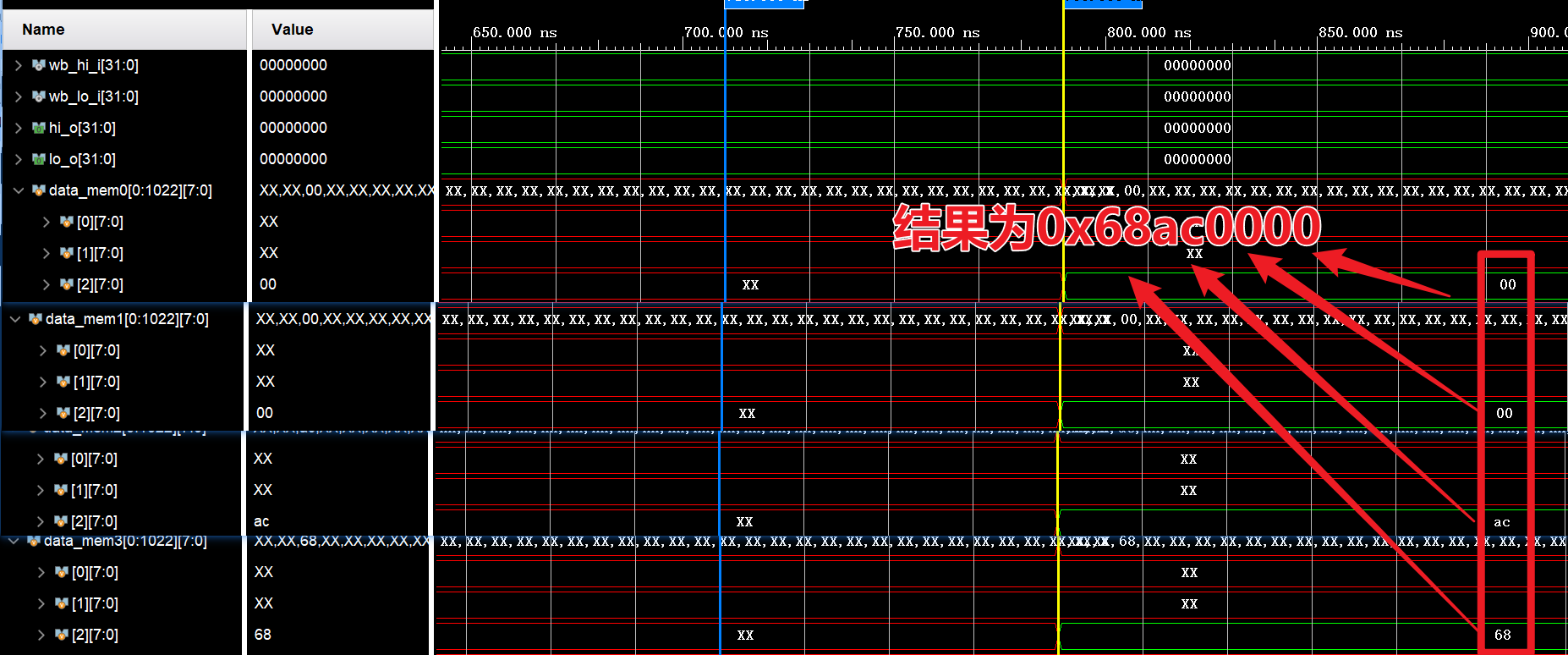
具体的写寄存器结果如下图所示：



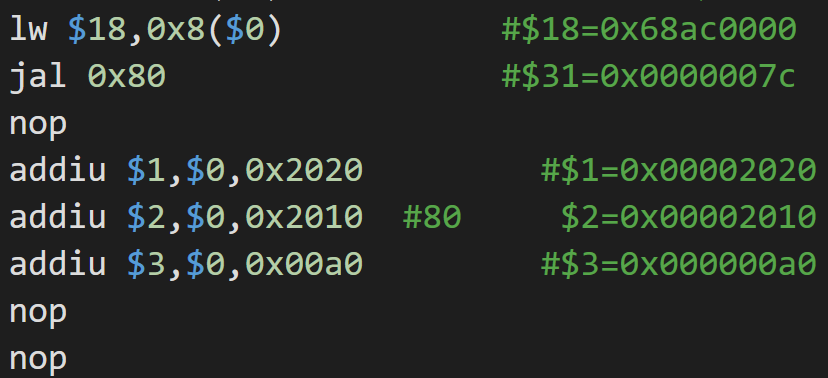
然后是一条访存指令：sw $14,0x8($0)。分析如下图所示：



在访存周期结束的时候写入data\_ram：

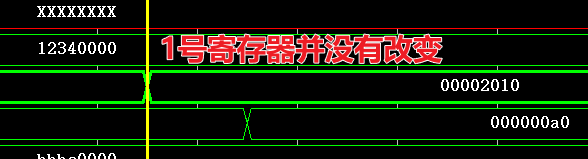


然后是一些算术指令，如下图所示：

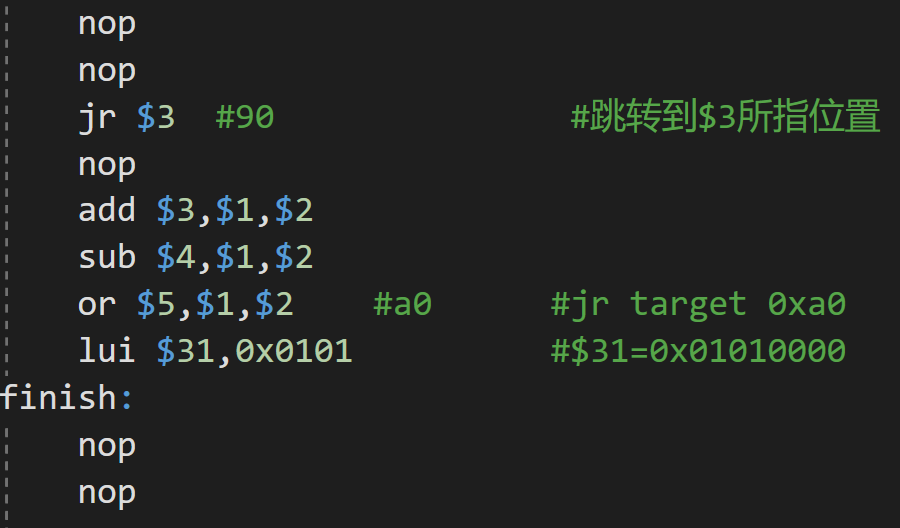


分别对应波形图：

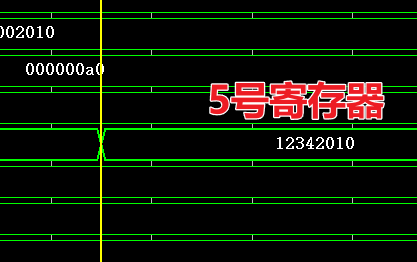




最后是这些指令：



执行结果：

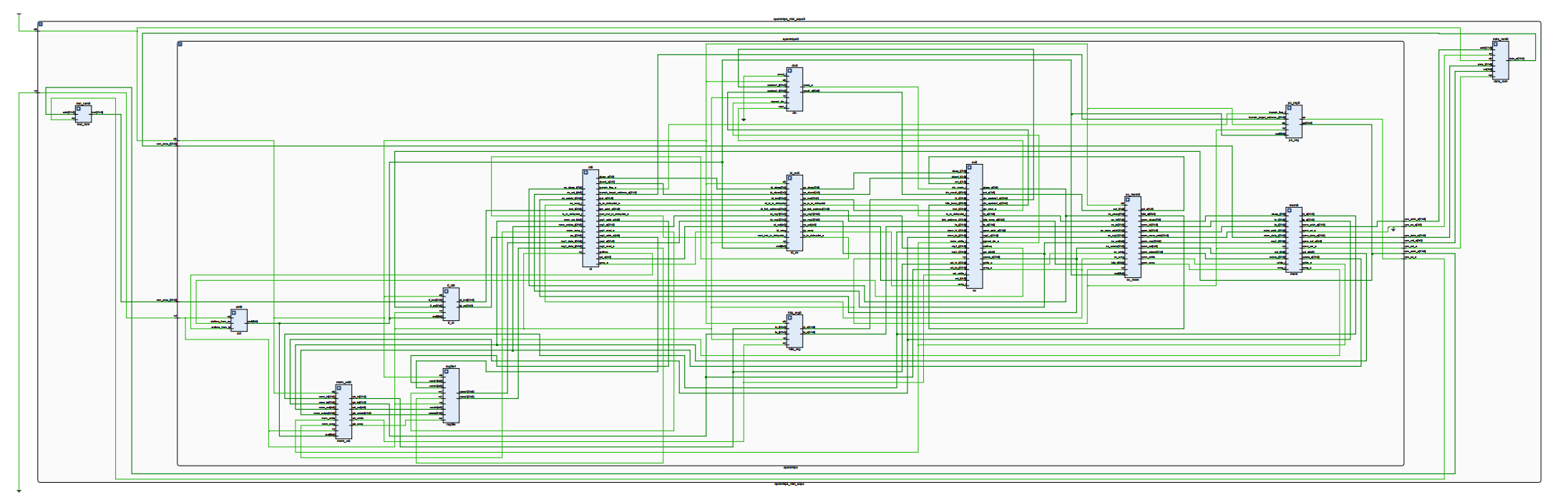




#### 测试用例2（func\_test2）

#### 添加功能测试

### 2. 五级流水CPU的RTL级逻辑电路



# 【总结】

# 【心得体会】

# 【参考文献（资料）】

[1]. 汪文祥,邢金璋.CPU设计实战.机械工业出版社.2021.

[2]. 雷思磊.自己动手写CPU.电子工业出版社.2014.

[3]. 高小鹏,计算机组成与实现,高等教育出版社,2018

[4]. 袁静波等，《计算机组成与结构》，机械工业出版社，2011

[5]. 王爱英，《计算机组成与结构》，清华大学出版社，2013

[6]. 白中英，《计算机组成原理》，科学出版社，2008