# 计算机组成原理大作业

——五级流水线CPU

**指导教师：熊波涛**

**小组成员：**

**目录**

[计算机组成原理大作业 1](#_Toc57223352)

[一、设计目的与要求 3](#_Toc57223353)

[1.1 实验内容 3](#_Toc57223354)

[1.2 实验要求 3](#_Toc57223355)

[二、CPU逻辑设计总体方案 3](#_Toc57223356)

[2.1 数据通路 3](#_Toc57223357)

[2.2 MIPS指令格式 5](#_Toc57223358)

[2.3 流水线CPU结构设计图 6](#_Toc57223359)

[2.4 流水线CPU逻辑流程图 6](#_Toc57223360)

[三、冒险解决策略 7](#_Toc57223361)

[3.1 数据冒险 7](#_Toc57223362)

[3.2 控制冒险 9](#_Toc57223363)

[四、 模块详细设计 9](#_Toc57223364)

[4.1取指令部分（IF） 9](#_Toc57223365)

[4.2指令译码部分（ID） 12](#_Toc57223366)

[4.3 执行部分（EX） 16](#_Toc57223367)

[4.4存储器访问部分（MEM） 19](#_Toc57223368)

[4.5寄存器堆写回部分（WB） 20](#_Toc57223369)

[4.6 顶层模块 21](#_Toc57223370)

[4.7 仿真模块 22](#_Toc57223371)

[五、 仿真模拟分析 22](#_Toc57223372)

[5.1 仿真波形图 22](#_Toc57223373)

[5.2阻塞分析 22](#_Toc57223374)

[六、 结论和体会 23](#_Toc57223375)

[6.1对本实验过程及方法、手段的改进建议 23](#_Toc57223376)

# 一、设计目的与要求

## 1.1 实验内容

1. 实验所需指令的查询及实现方法
2. 对于CPU的基本构造和功能实现有了深入的了解
3. 基本功能部件的设计与实现。
4. 掌握流水线MIPS微处理器的工作原理。
5. 了解数据冒险、控制冒险的概念、发生原因以及解决方法
6. CPU各主要功能部件的设计与实现。
7. CPU的封装。
8. 掌握流水线MIPS微处理器的测试方法。

## 1.2 实验要求

1. 至少支持add、sub、and、or、lw、sw、beq、bne八条指令。
2. 采用5级流水线技术，具有数据前推机制。

# 二、CPU逻辑设计总体方案

流水线是数字系统中一种提高系统稳定性和工作速度的方法，广泛应用于高档CPU的架构中。根据MIPS处理器的特点，将整体的处理过程分为取指令（IF）、指令译码（ID）、执行（EX）、存储器访问（MEM）和寄存器会写（WB）五级，对应多周期的五个处理阶段。一个指令的执行需要5个时钟周期，每个时钟周期的上升沿来临时，此指令所代表的一系列数据和控制信息将转移到下一级处理。  

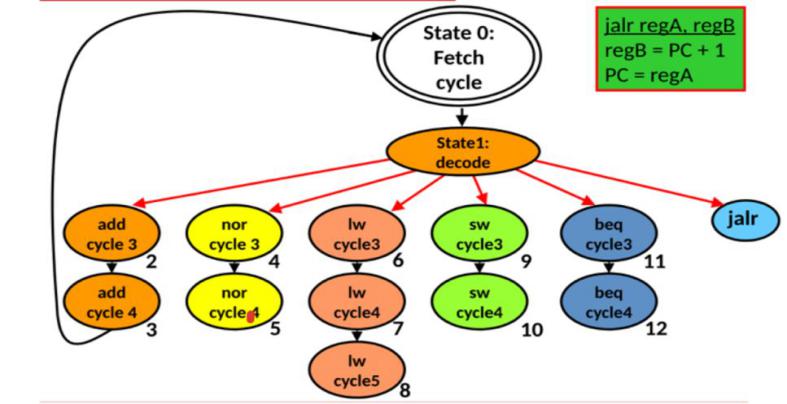
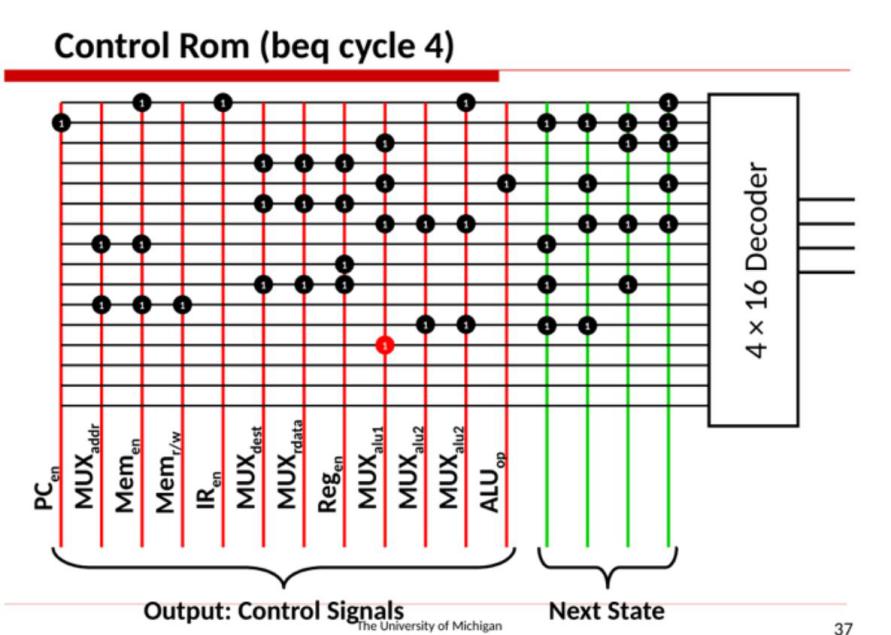

## 2.1 数据通路

（1） IF级：取指令部分。  
包括指令储存器和PC寄存器及其更新模块，负责根据PC寄存器的值从指令存储器中取出指令编码和对PC的值进行更新。

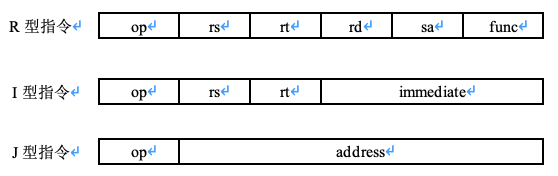
（2） ID级：指令译码部分。  
根据独处的指令编码形成控制信号和读寄存器堆输出的寄存器的值。  
流水线冒险检测也在该级进行，冒险检测电路需要上一条指令的MemRead，即在检测到冒险条件成立时，冒险检测电路产生stall信号清空ID/EX寄存器，插入一个流水线气泡。

（3） EX级：执行部分。  
根据指令的编码进行算数或者逻辑运算或者计算条件分支指令的跳转目标地址。  
此外LW、SW指令所用的RAM访问地址也是在本级上实现。控制信号有ALUCode、ALUSrcA、ALUScrB和RegDst，根据这些信号确定ALU操作、选择两个ALU操作数A、B，并确定目标寄存器。另外，数据转发也在该级完成。数据转发控制电路产生ForwardA和ForwardB两组控制信号。

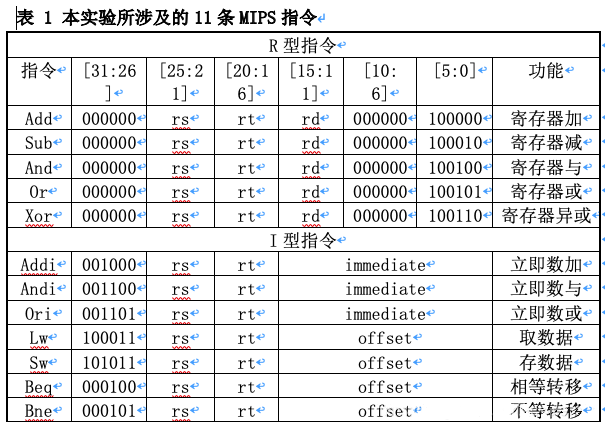
（4） MEM级：存储器访问部分。  
只有在执行LW、SW指令时才对存储器进行读写，对其他指令只起到一个周期的作用。该级只需存储器写操作允许信号MemWrite。

（5） WB级：寄存器堆写回部分。  
该级把指令执行的结果回写到寄存器文件中。  
该级设置信号MemtoReg和寄存器写操作允许信号RegWrite，其中MemtoReg决定写入寄存器的数据来自于MEM级上的缓冲值或来自于MEM级上的存储器。  


## 2.2 MIPS指令格式

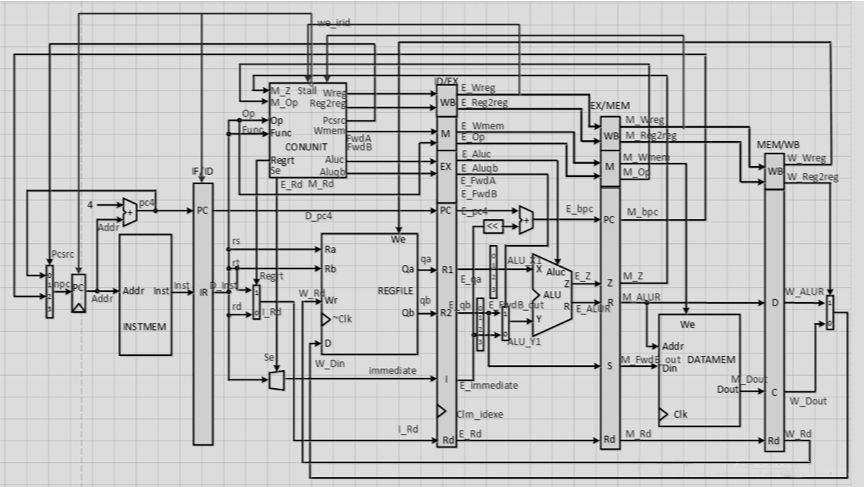
MIPS指令系统结构有MIPS-32和MIPS-64两种。本实验的MIPS指令选用MIPS-32。以下所说的MIPS指令均指MIPS-32。  
MIPS的指令格式为32位。下图给出了MIPS指令的3种格式。  


本实验只选取了11条典型的MIPS指令来描述CPU逻辑电路的设计方法。下图列出了本实验的所涉及到的11条MIPS指令。

  
R型指令的op均为0，具体操作由func指定。rs和rt是源寄存器号，rd是目的寄存器号。移位指令中使用sa指定移位位数。

I型指令的低16位是立即数，计算时需扩展到32位，依指令的不同需进行零扩展和符号扩展。

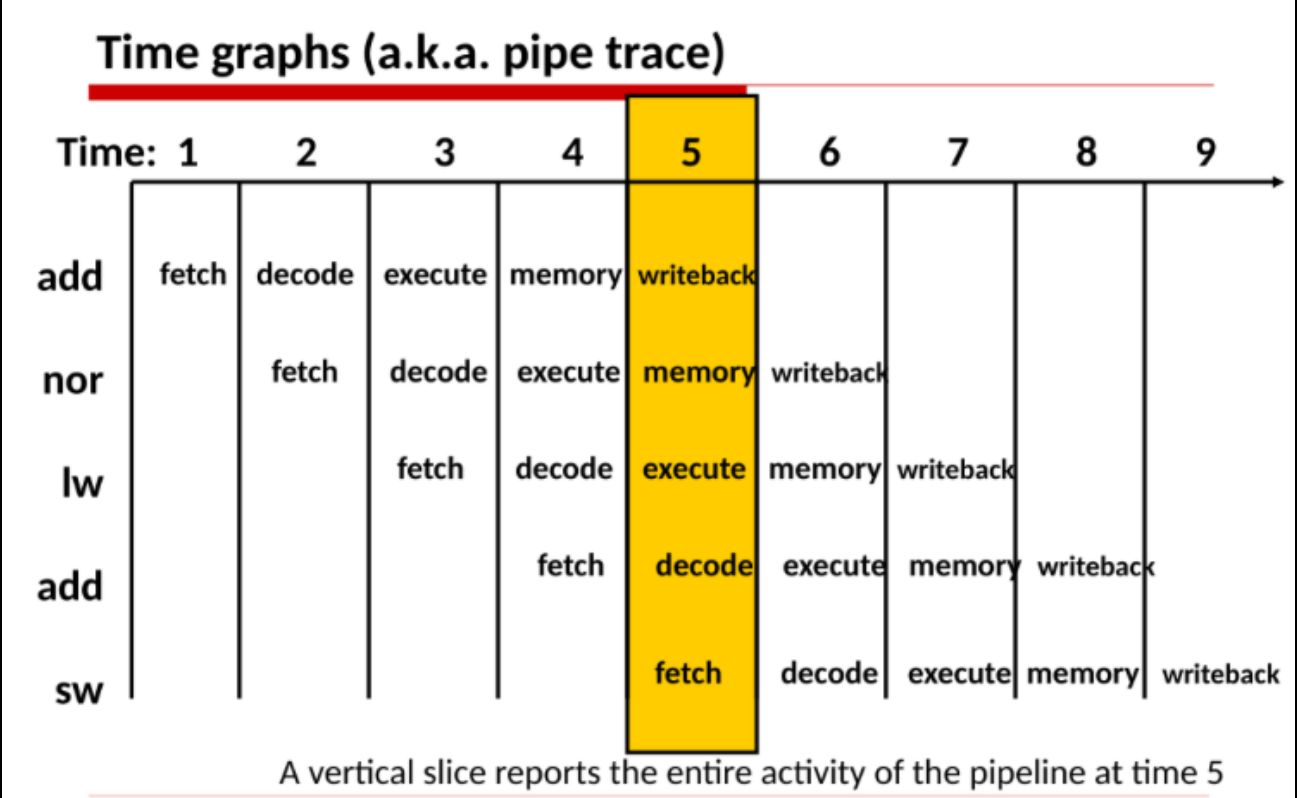
## 2.3 流水线CPU结构设计图



上图是一个简单的基本上能够在流水线上完成所要求设计的指令功能的数据通路和必要的控制线路图。其中指令储存在指令储存器，数据储存在数据存储器。而且实现了数据冒险和控制冒险的解决方案。但是此结构仅解决了数据冒险，没有解决控制冒险，若要想解决控制冒险，需要把EX级的移位器和计算分支目标地址的加法器前移到ID。在下文中会给出具体的解决方法。

## 2.4 流水线CPU逻辑流程图

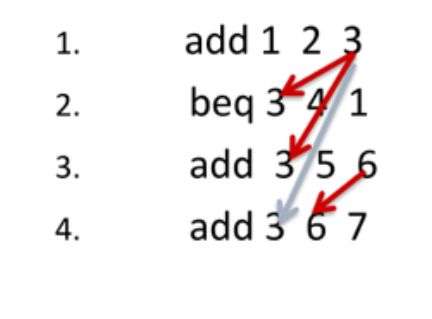
根据实验原理中的流水线CPU总体结构图，我们可以清楚的知道流水线CPU的设计应包括五级。其中为了运行整个CPU还需要加入一个顶层模块（MAIN）来调用这些模块，所以自然地，这些模块为顶层模块的子模块。设计流程逻辑图如下（左边为流水线寄存器组，右边为各级模块）。



# 三、冒险解决策略

## 3.1 数据冒险

在基本流水线中相邻两条指令的前一条指令还没有更新目的寄存器时，后一条指令就已经先读取了该寄存器的旧值，使得指令的计算结果出现错误。这样相关的问题就称为数据冒险。下面给出本实验所使用的解决方法。



### 3.1.1 寄存器堆的写操作提前半个时钟周期

在未提前半周期时CPU中，指令在时钟周期结束时的上升沿将ID级寄存器的值锁存进ID/EX，或将ALU的计算结果更新寄存器的值。但是寄存器的读、写的操作时间实际上只有时钟周期的一半。因此可以把寄存器堆的写操作提前到时钟周期中间的下降沿完成。那么后半个时钟周期就可以将写入之后的值读出。这样做后，同一级的数据冒险得到解决。

### 3.1.2内部前推

运算指令的结果在EX级结束时，就已经锁存在EX/MEM的Rd中，然而寄存器的值在ID结束时就已经锁存在ID/EX，但在EX级才真正使用这些值。  
所以为了支持内部前推，需在ALU的两个输入端之前，分别增加一个多路选择器和相应的数据通路，并检测处于EX级指令的两个源操作寄存器号是否和处于MEM级或WB级指令的目的寄存器号相等。下面给出检测条件。

1. [条件a] E\_Rs == EX/MEM.Rd，  
   **判断EX级指令的rs字段是否和MEM级指令的目的寄存器号相同**
2. [条件b] E\_Rs == MEM/WB.Rd，  
   **判断EX级指令的rs字段是否和WB级指令的目的寄存器号相同**
3. [条件c] (E\_Rt == EX/MEM.Rd) & ((E\_Inst == I\_add) | (E\_Inst == I\_sub) | (E\_Inst == I\_and) | (E\_Inst == I\_or) | (E\_Inst == sw) | (E\_Inst == beq) | (E\_Inst == bne))  
   **判断EX级指令的rt字段是否和MEM级指令的目的寄存器号相同**
4. [条件d] (E\_Rt == MEM/WB.Rd) & ((E\_Inst == I\_add) | (E\_Inst == I\_sub) | (E\_Inst == I\_and) | (E\_Inst == I\_or) | (E\_Inst == sw) | (E\_Inst == beq) | (E\_Inst == bne))  
   **判断EX级指令的rt字段是否和WB级指令的目的寄存器号相同**

**同时应该考虑以下几种特殊情况。**

1. 某些指令可能不写回寄存器，例如sw和beq指令，或者某些指令的写信号被关闭。所以还需检测处于MEM级或WB级指令的寄存器堆写使能信号M\_Wreg或W\_Wreg是否有效。
2. 寄存器$0始终为0，不必考虑在$0上产生的数据冒险，即第三条指令分别与第一条、第二条指令存在数据冒险。按照执行逻辑，当第三条指令处于EX级时应选择处于MEM级的第二条指令的前推。而不能选择第一条前推。所以在判断逻辑模块的代码实现时，应先判断相邻两条指令是否存在数据冒险。



### 3.1.3 lw指令的数据冒险

内部前推有显而易见的局限性，因为内部前推必须要求前一条指令在EX结束时更新，但是LW指令最早只能在WB级读出寄存器的值。因此无法及时提供给下一条指令的EX级使用。  
分析流水线时序图，可以发现lw指令的下一条指令，需要阻塞一个时钟周期，才能确保该指令能获得正确的操作数值，下面给出具体解决方法。  
检测是否存在lw指令的数据冒险。  
检测单元仍放置于CONUNIT部件内，且Reg2reg信号可以唯一区分lw指令和其他指令。检测条件可写为：  
((Rs == E\_Rd) | (Rt == E\_Rd)) & (E\_Reg2reg == 0) & (E\_Rd != 0) & (E\_Wreg == 1)

**暂停流水线的实现：**  
我们可以通过插入气泡来暂停流水线，也即是关闭PC寄存器和IF/ID流水线寄存器组的写使能信号，而且将ID/EX流水线寄存器组的Clrn端口信号清零。

## 3.2 控制冒险

在处理beq和bne指令时，条件分支指令或者跳转指令的后续指令有可能在目标地址形成之前或分支条件形成之前就已经进入流水线。这样相关的问题就称为控制冒险。下面给出本实验所使用的解决方法。

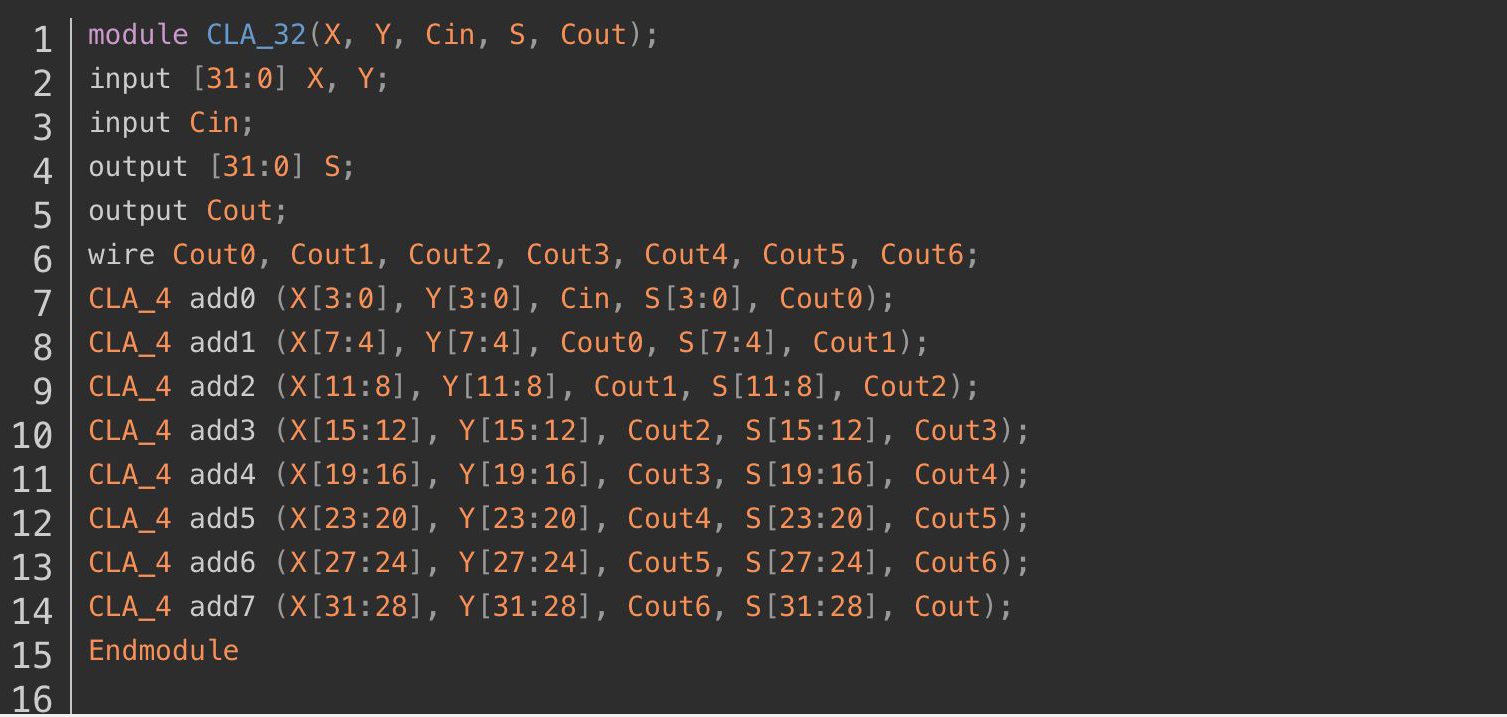
# 四、 模块详细设计

## 4.1取指令部分（IF）

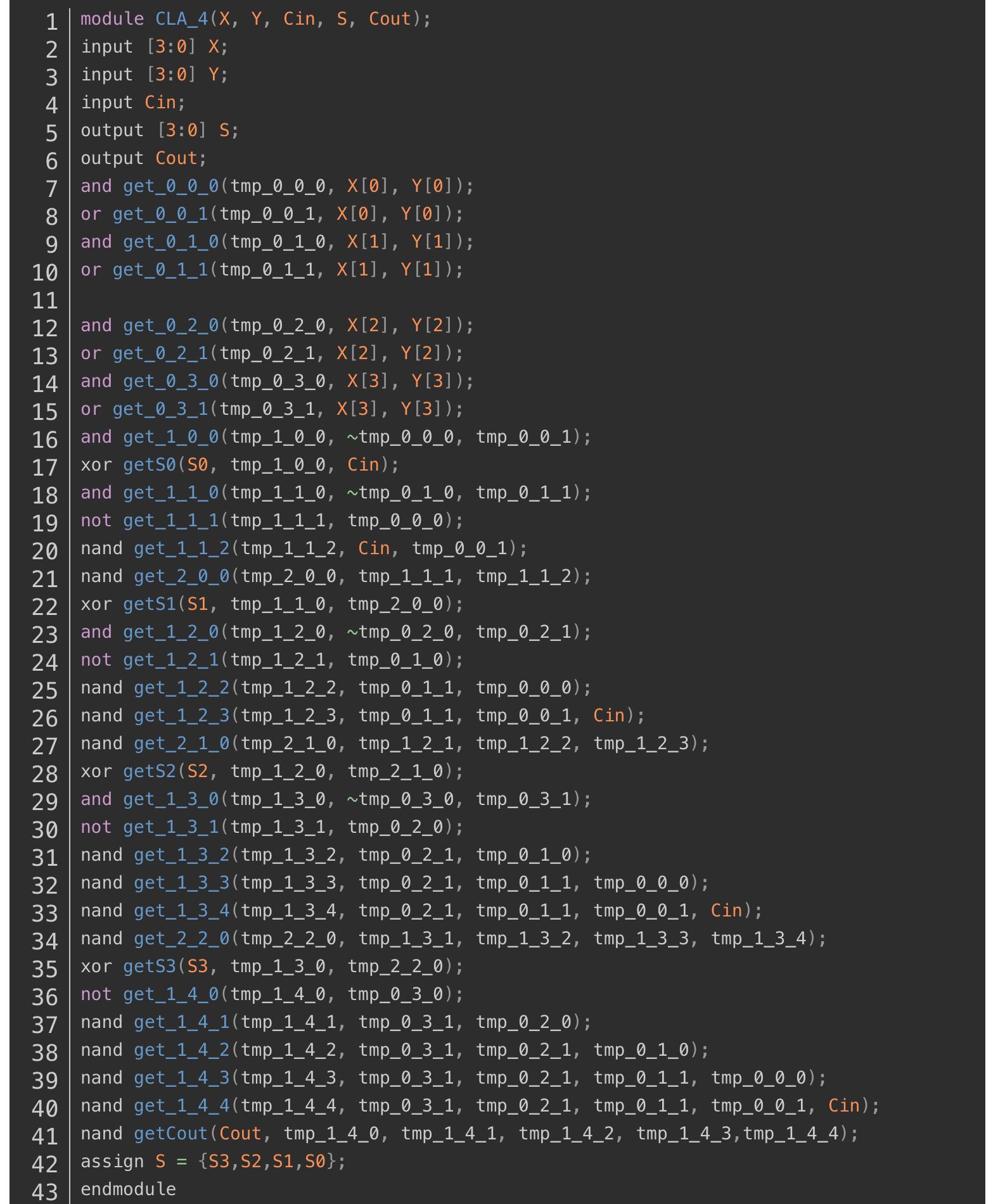
**模块功能**  
作为PC寄存器的更新信号。



1. CLA\_32



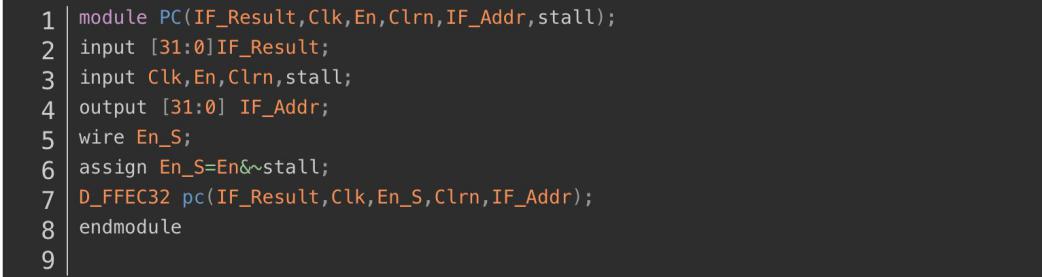
2. CLA\_4



### 4.1.2 PC

**模块功能**  
用于给出指令在指令储存器中的地址，且当发生数据冒险时，需要保持PC寄存器不变。

**主要实现代码**



### 4.1.3 INSTMEM

**模块功能**  
依据当前pc，读取指令寄存器中相对应地址Addr[6:2]的指令。

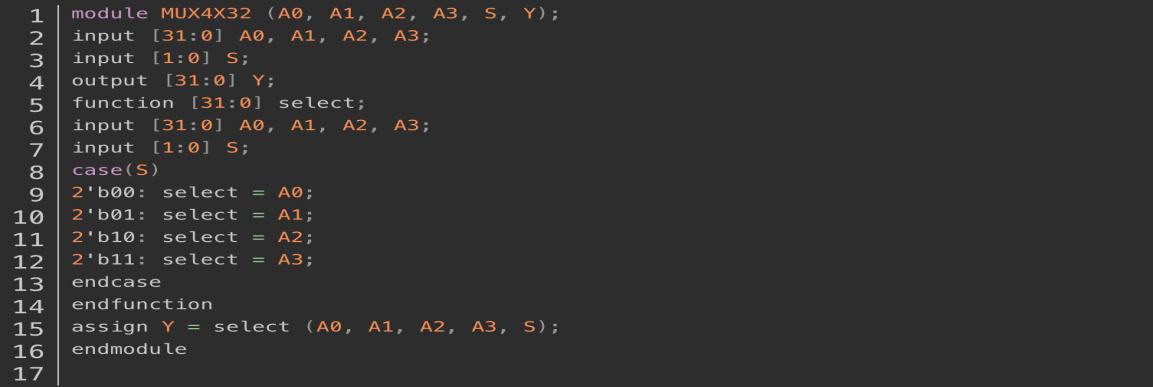
**主要实现代码**

****

### 4.1.4 MUX4X32

**模块功能**  
实现目标地址的选择

**主要实现代码**



### 4.1.5 REG\_ifid

**模块功能**  
寄存IF级的输出指令，分割IF级和ID级的指令或控制信号，防止相互干扰，在IF级执行结束时将指令的控制信号传递至下一级。

**主要实现代码**

****

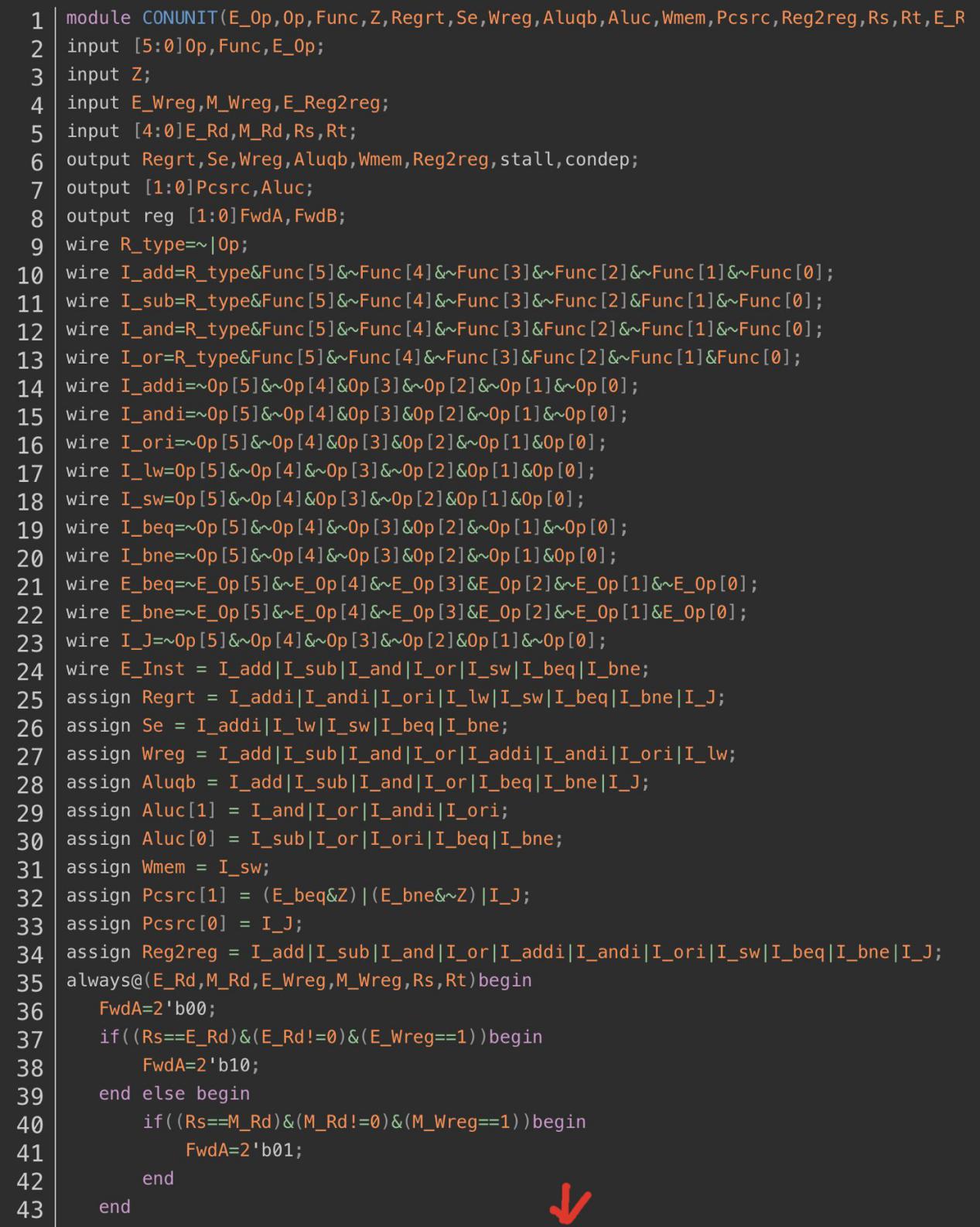
## 4.2指令译码部分（ID）

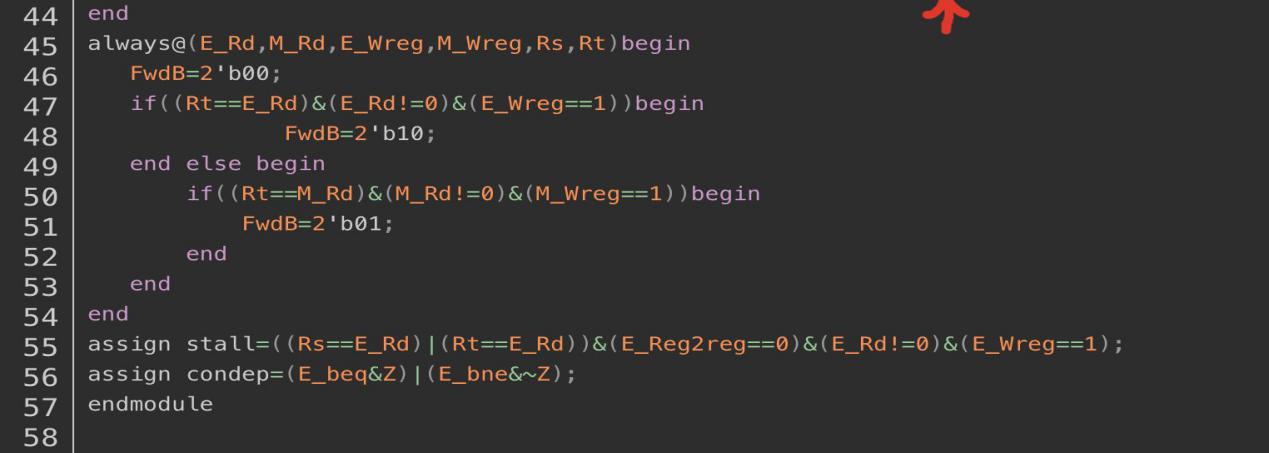
### 4.2.1 CONUNIT

**模块功能**  
控制器是作为CPU控制信号产生的器件，通过通过解析op得到该指令的各种控制信号，使其他器件有效或无效。

**实现思路**  
在单周期CONUNIT的基础上，需要增加一些功能，如判断处于ID级的指令和处于EX级或MEM级的指令是否存在数据冒险，是否存在Lw数据冒险，是否存在控制冒险。

**主要实现代码**

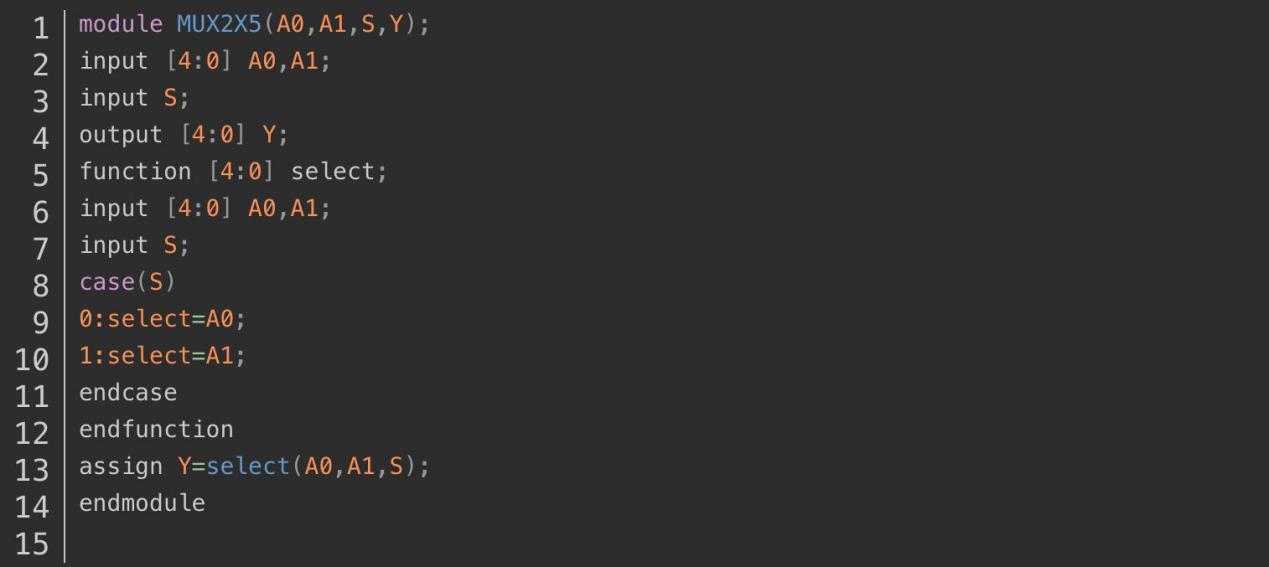
****

****

### 4.2.2 MUX2X5

**模块功能**  
R型指令和I行指令的Wr信号不同，所以需要一个5位二选一选择器进行  
选择。

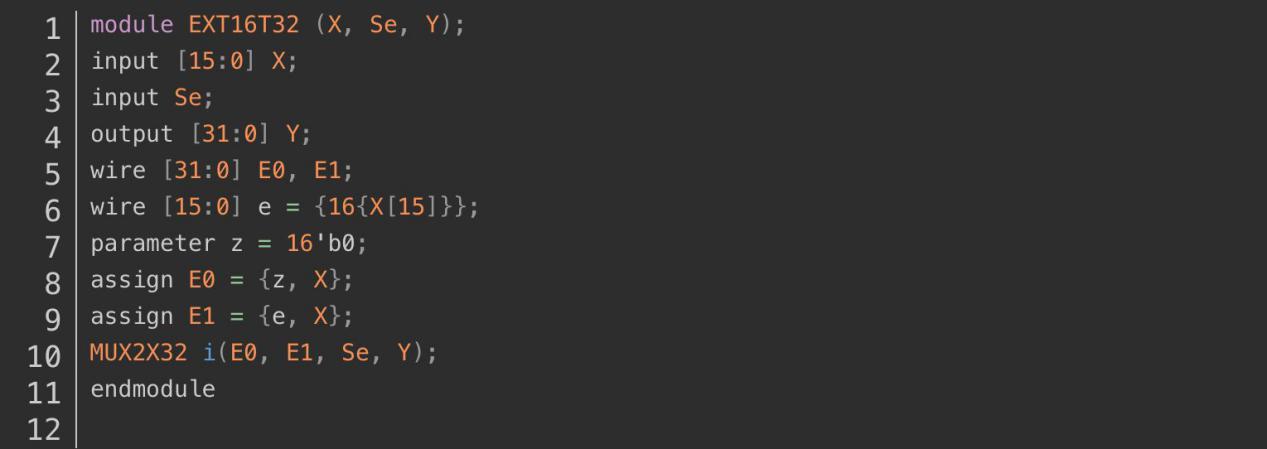
**主要实现代码**

****

### 4.2.3 EXT16T32

**模块功能**  
I指令的addi需要对立即数进行符号拓展，andi和ori需要对立即数进行零扩展，所以需要一个扩展模块。

**主要实现代码**

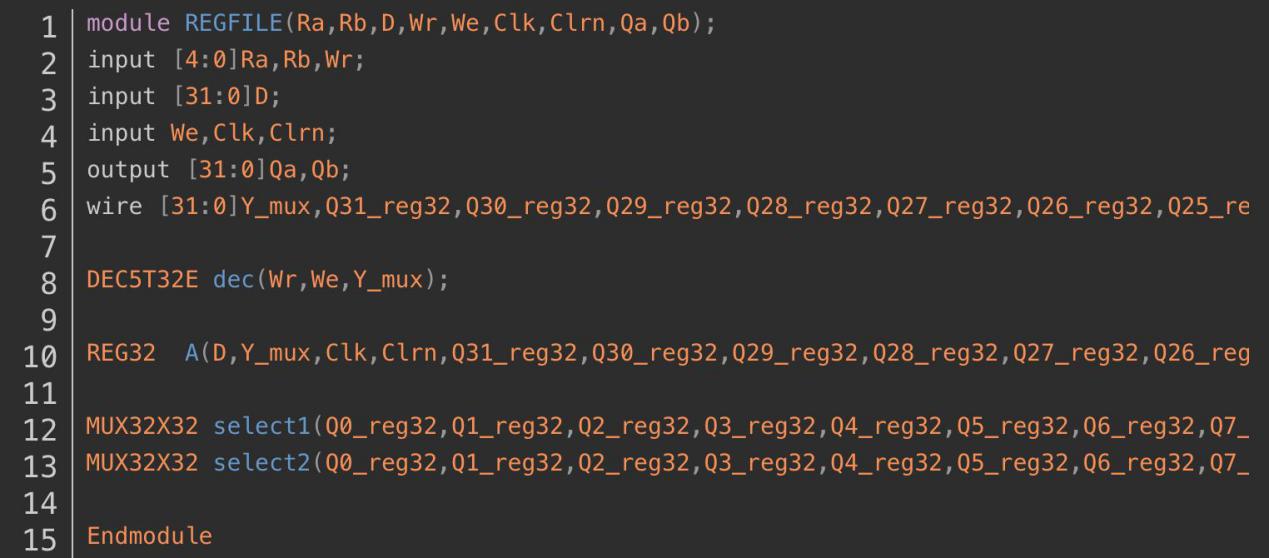
****

### 4.2.4 REGFILE

**所处位置**

**模块功能**  
给出要读取的两个寄存器编号和要写入的寄存器编号，然后由Qa和Qb端口更新Ra和Rb端口的输入编号分别输入其值。

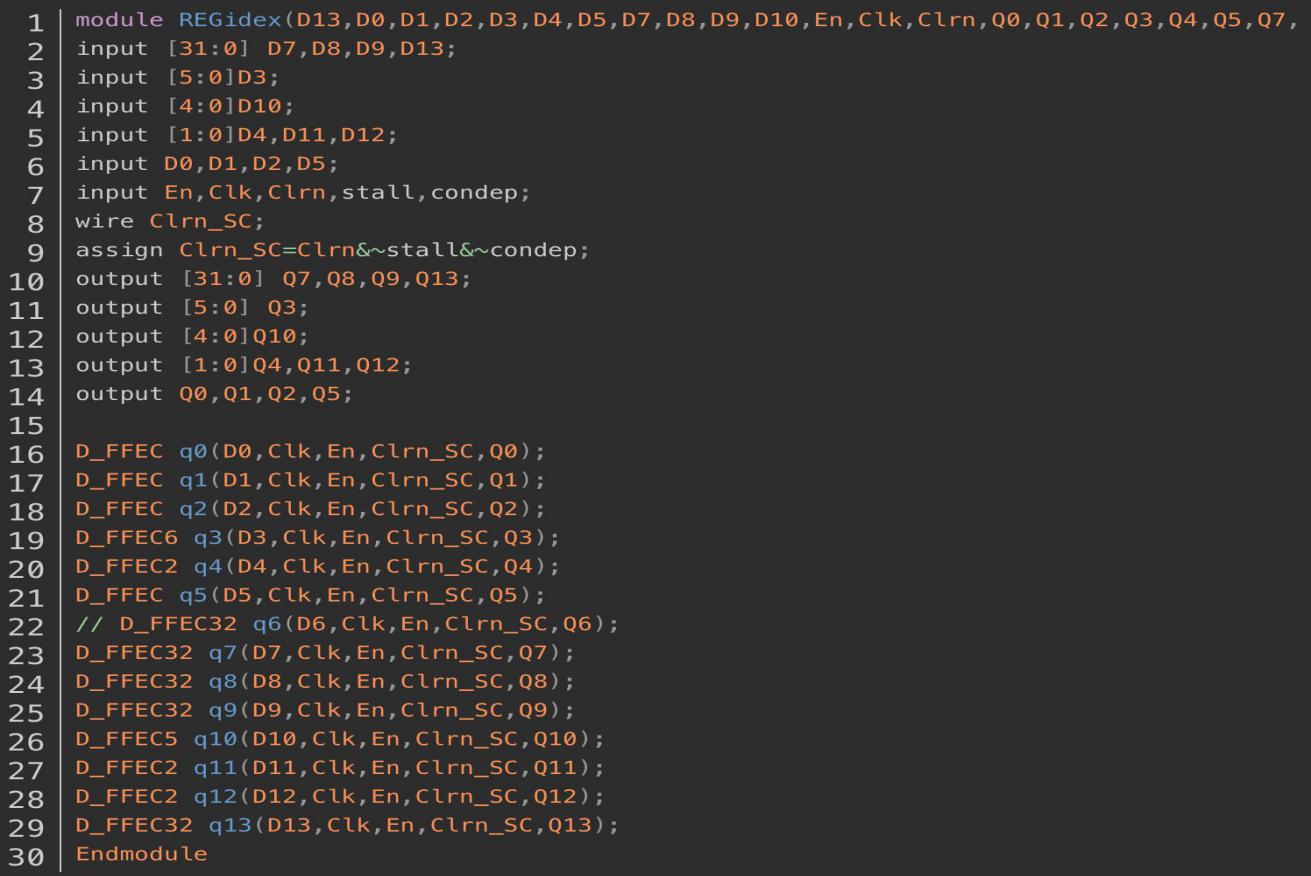
**主要实现代码**



### 4.2.5 REG\_idex

**模块功能**  
寄存ID级的输出指令，分割ID级和EX级的指令或控制信号，防止相互干扰，在ID级执行结束时将指令的控制信号传递至下一级。

**主要实现代码**

****

## 4.3 执行部分（EX）

**4.3.1 SHIFTER32\_L2**

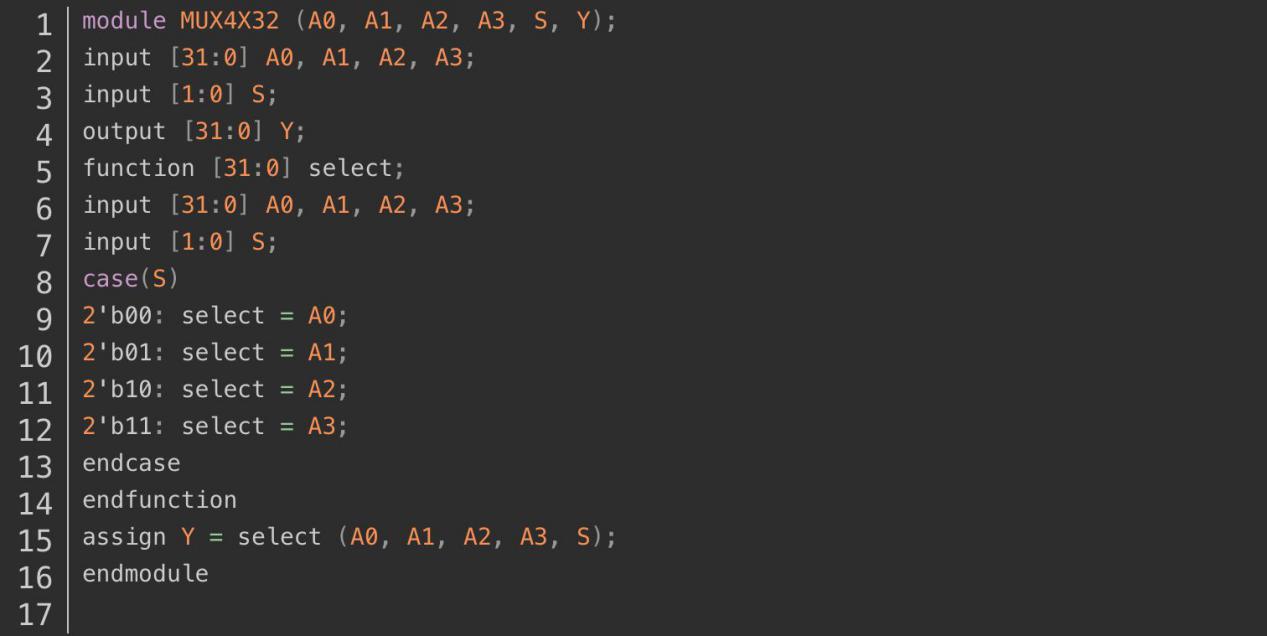
**模块功能**  
使用32位移位器SHIFTER32，固定左移两位即可

**主要实现代码**

****

### 4.3.2 MUX4X32

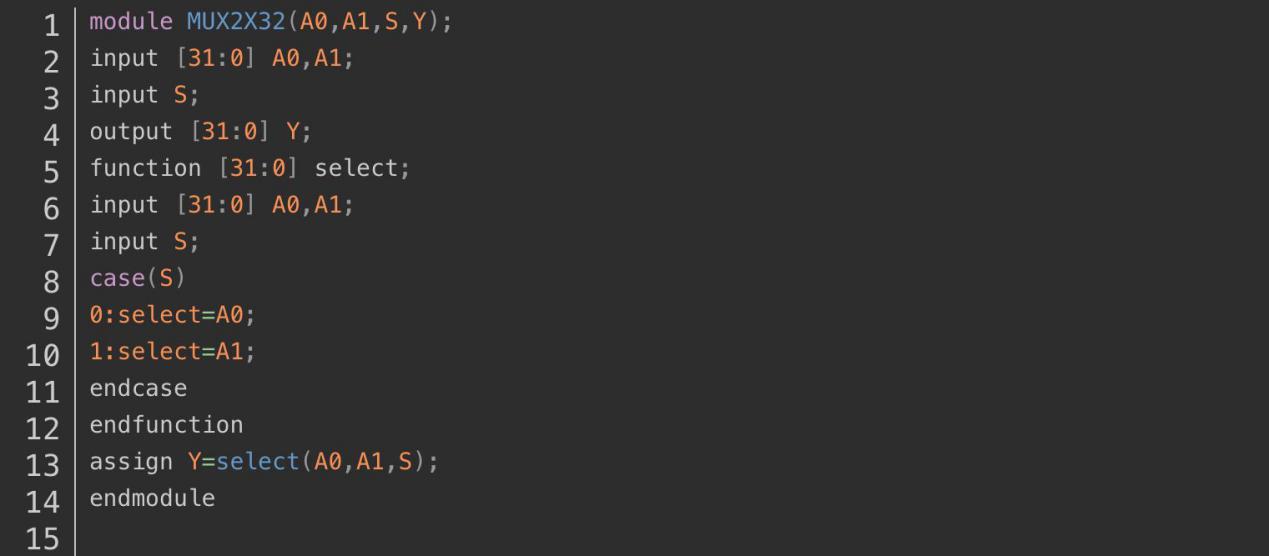
**模块功能**  
在内部前推的工作中，ALU的两个输入端都需要选择来自ID/EX寄存器或者EX/MEM寄存器或者MEM/WB寄存器所锁存的值和立即数做选择，后输作为ALU中Y端的信号，输出信号。  


****

### 4.3.3 MUX2X32

**模块功能**  
ALU的Y端输入信号种类根据指令的不同而不同

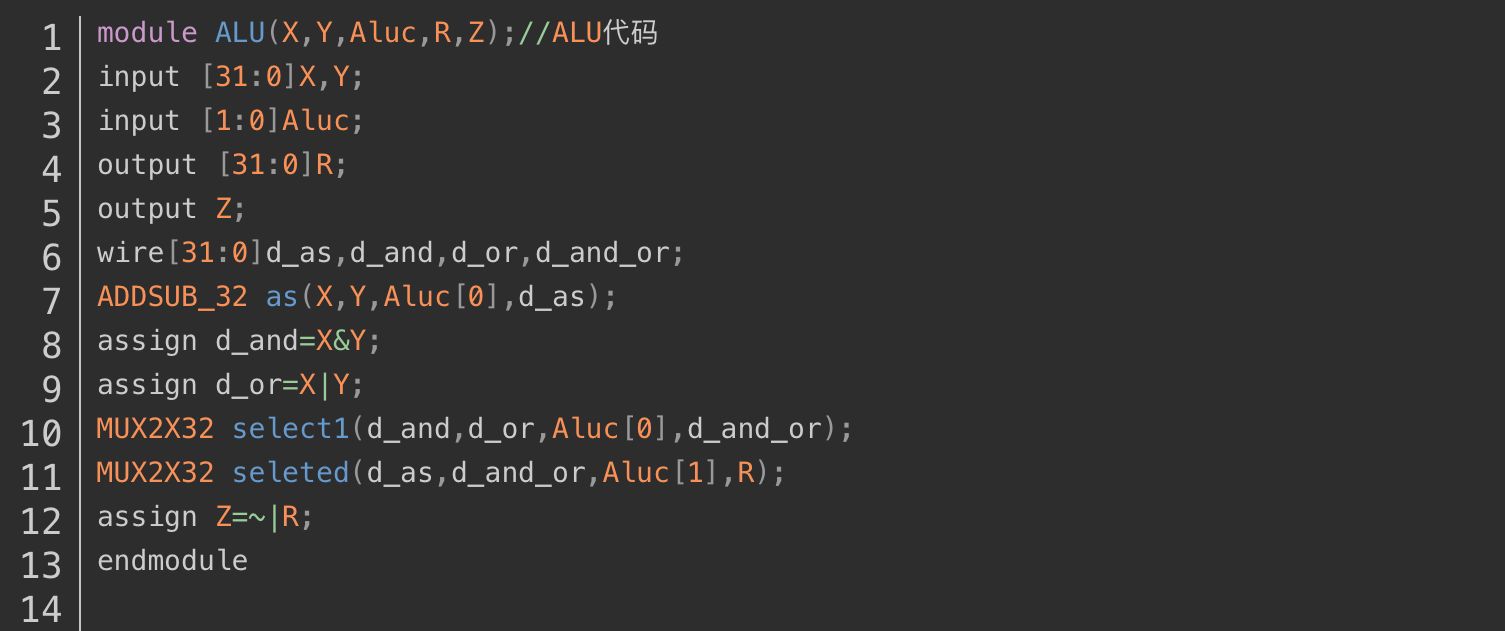
**主要实现代码**

****

### 4.3.4 ALU

**模块功能**  
算数逻辑部件，需要实现加，减，按位与，按位或。

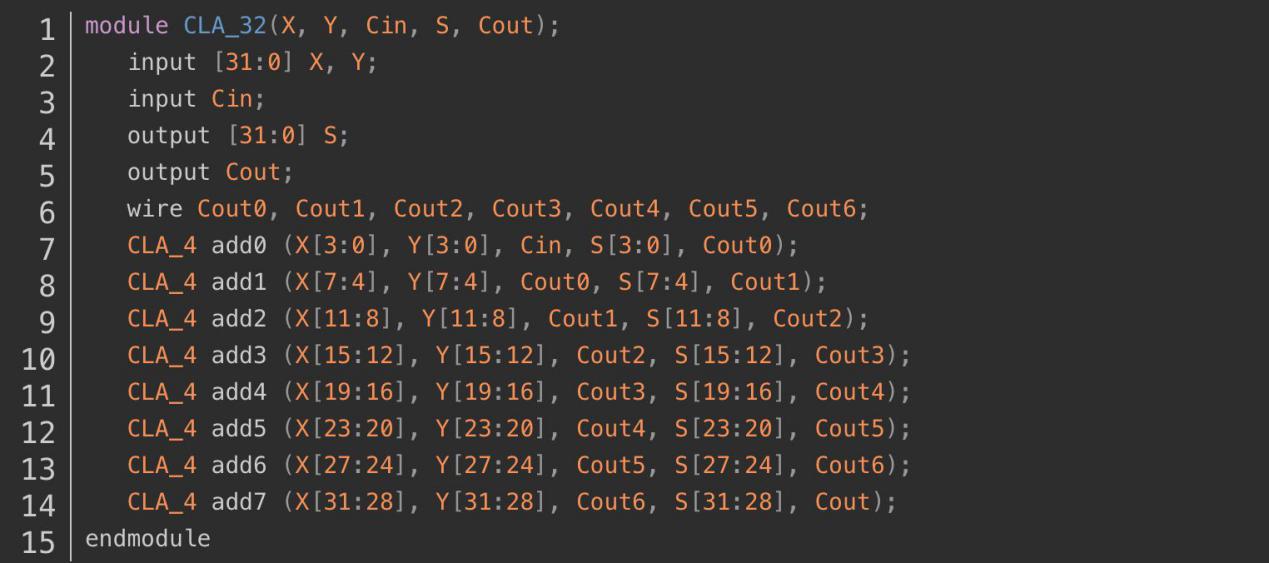
**主要实现代码**



### 4.3.5 CLA\_32

**模块功能**  
在beq和bne指令时如果发生地址的跳转，需要使用32位加法器，跳转地址是pc+4的地址和立即数之和的结果。

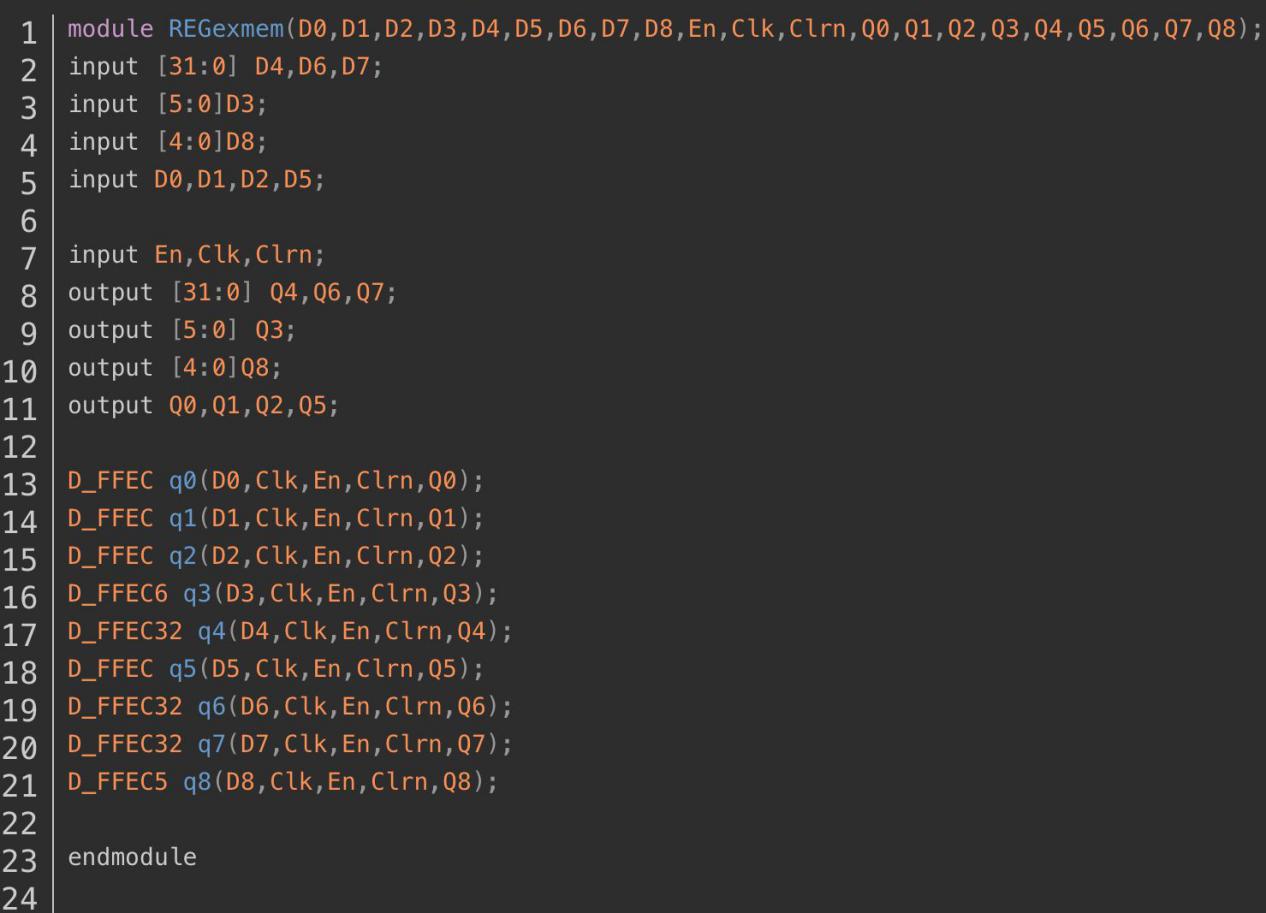
**主要实现代码**

****

### 4.3.6 REG\_exmem

**模块功能**  
寄存EX级的输出指令，分割EX级和MEM级的指令或控制信号，防止相互干扰，在EX级执行结束时将指令的控制信号传递至下一级。

**主要实现代码**

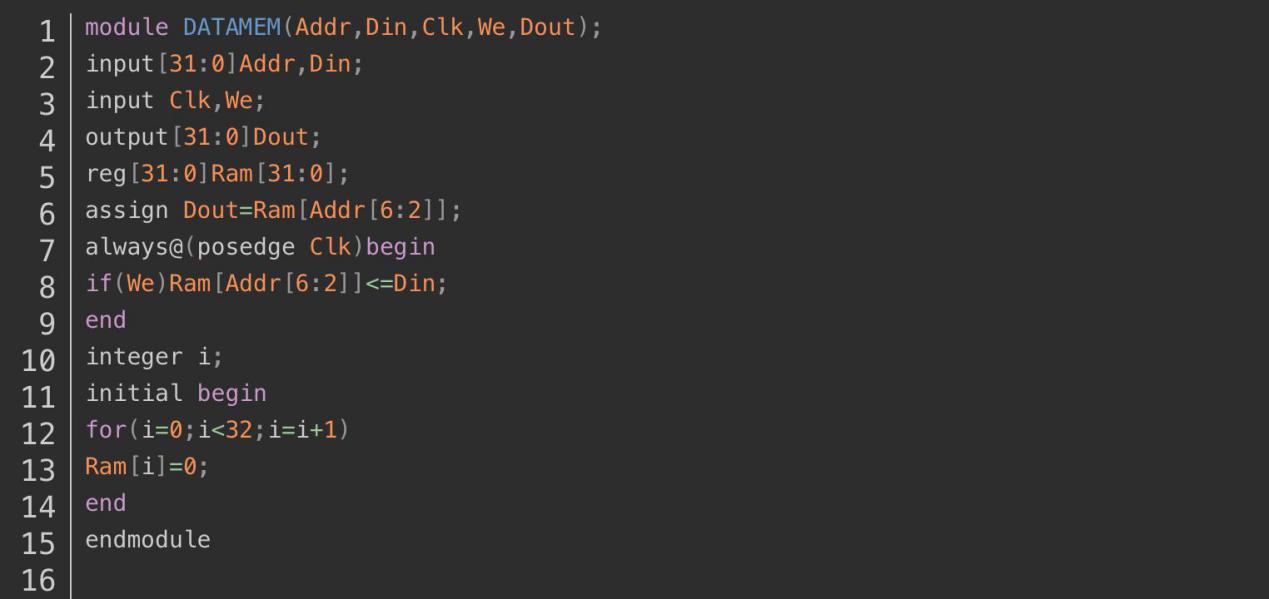
****

## 4.4存储器访问部分（MEM）

### 4.4.1 DATAMEMß

**模块功能**  
数据存储器，通过控制信号，对数据寄存器进行读或者写操作，并且此处模块额外合并了输出DB的数据选择器，此模块同时输出写回寄存器组的数据DB。

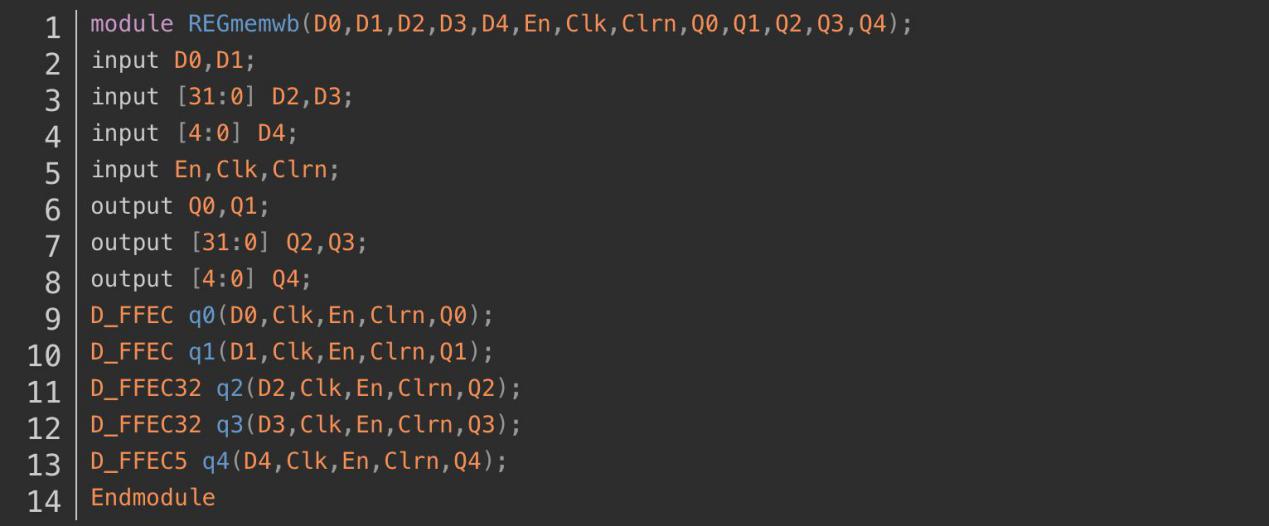
**主要实现代码**



### 4.4.2 REG\_memwb

**模块功能**  
数据存储器，通过控制信号，对数据寄存器进行读或者写操作，并且此处模块额外合并了输出DB的数据选择器，此模块同时输出写回寄存器组的数据DB。

**主要实现代码**

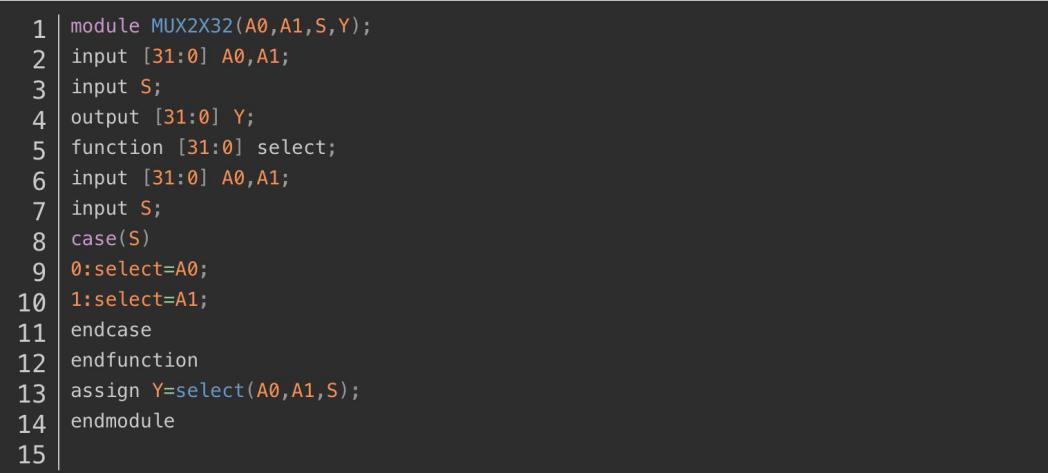


## 4.5寄存器堆写回部分（WB）

### 4.5.1 MUX2X32

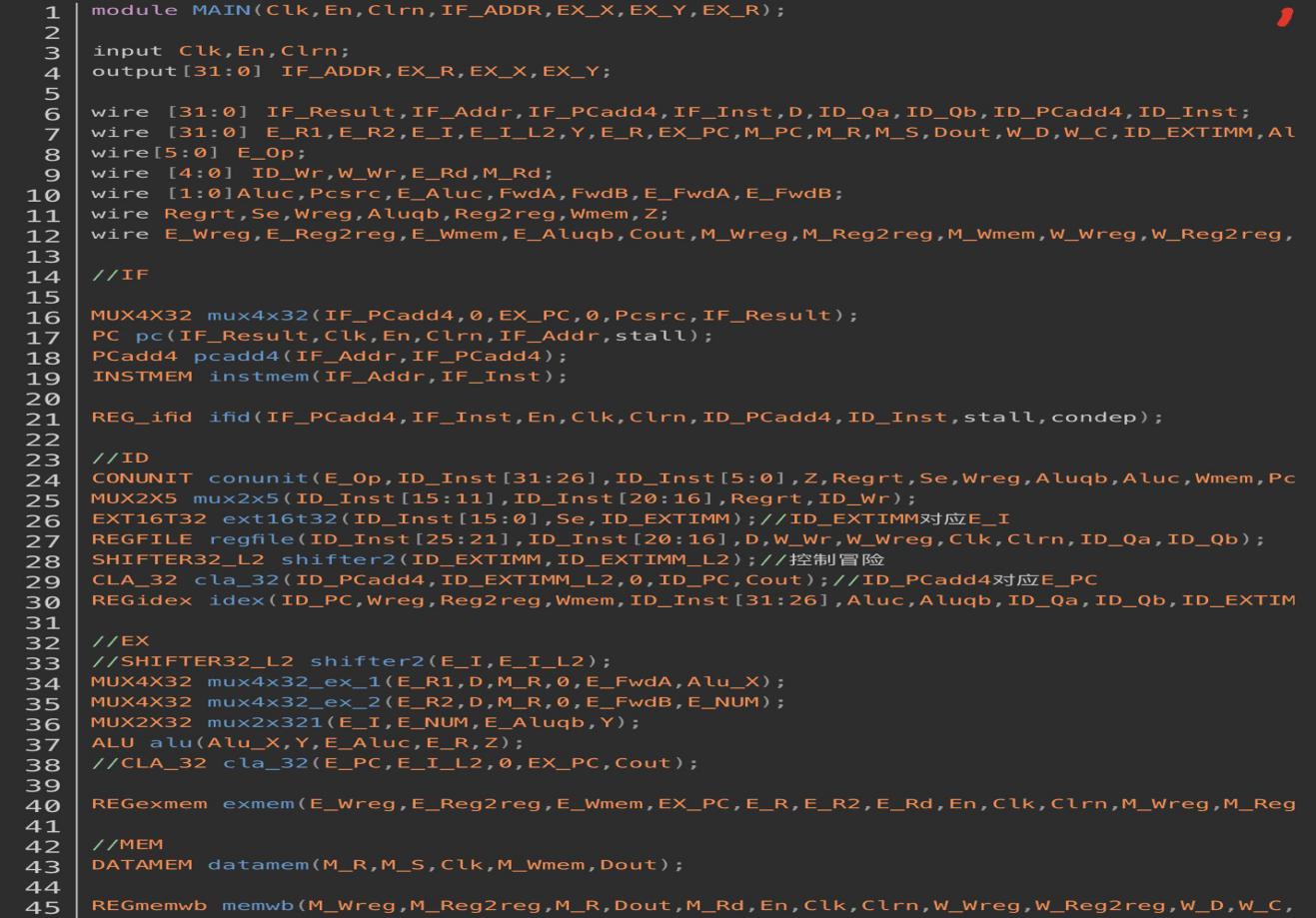
**模块功能**  
对写入寄存器的数据进行选择

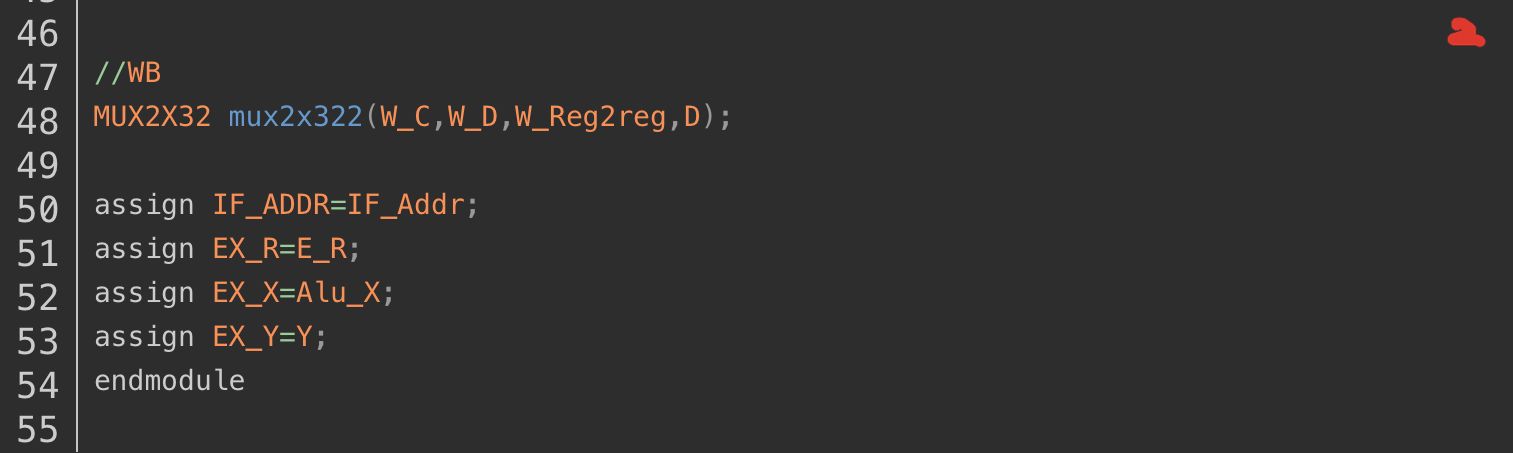
主要实现代码



## 4.6 顶层模块

模块功能  
实现CPU的封装，设计输出信号使得在方正时便于观察其波形图

****

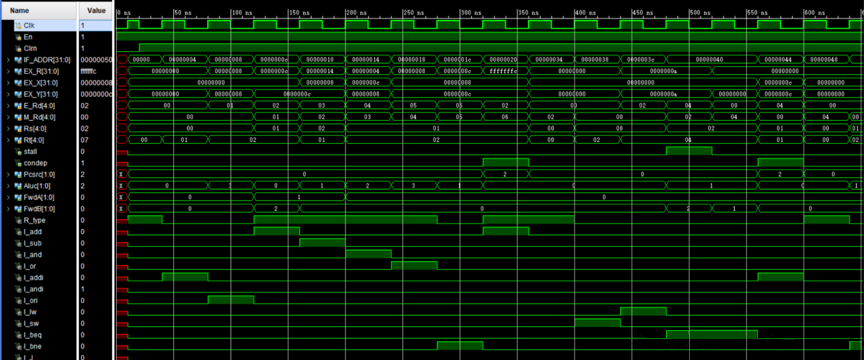
****

## 4.7 仿真模块

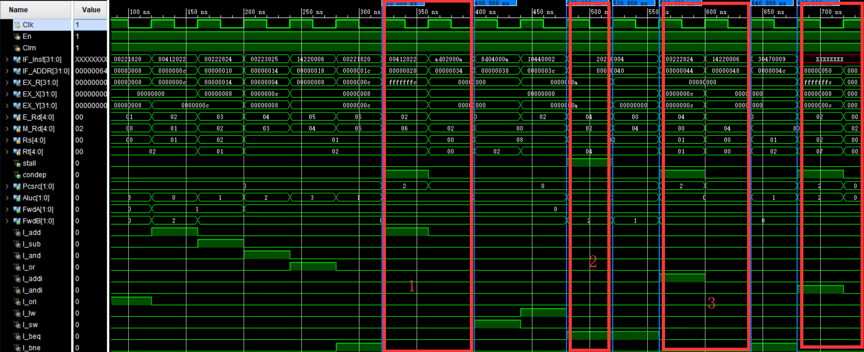
****

# 五、 仿真模拟分析

## 5.1 仿真波形图



## 5.2阻塞分析

  
第一个红框是流水线CPU因为bne结构冒险而暂停的两个时钟周期，  
第二个红框是流水线CPU因为lw数据冒险而暂停的一个时钟周期，  
第三个红框是流水线PCU因为beq数据冒险而暂停的两个时钟周期  
第四个红框是流水线PCU因为bne数据冒险而暂停的时钟周期。

# 六、 结论和体会

## 6.1对本实验过程及方法、手段的改进建议

1. 使用模块化设计，过程中，应该每个部件都分别进行调试之后再组装在一起，各部件尽量再拆分为更小的部件组合而成。
2. 使用流水线结构，提高CPU的运行效率

3.考虑数据冒险和控制冒险，可通过数据前传和空指令解决