成绩评定日期：

实验报告成绩：

2024～2025学年秋季学期

**《计算机系统》必修课**

课程实验报告



班级：人工智能2202班 人工智能2201班（未来实验班）

组长：刘庆然

组员：马智鸿

报告日期：2024.01.04

目录

[一、 分工及总体框架 6](#_Toc186924977)

[1.分工 6](#_Toc186924978)

[2.运行环境及所用工具 6](#_Toc186924979)

[3.整体框架 7](#_Toc186924980)

[1. 模块功能概述： 8](#_Toc186924981)

[Testbench (tb\_top) 8](#_Toc186924982)

[CPU\_Top (mycpu\_top) 8](#_Toc186924983)

[MMU (mmu0 和 mmu1) 8](#_Toc186924984)

[Core (mycpu\_core) 8](#_Toc186924985)

[流水线阶段子模块（IF、ID、EX、MEM、WB） 9](#_Toc186924986)

[控制模块 (CTRL) 9](#_Toc186924987)

[RF（寄存器文件） 10](#_Toc186924988)

[2. 模块之间的连接关系及其功能： 10](#_Toc186924989)

[Testbench (tb\_top) → CPU\_Top (mycpu\_top) 10](#_Toc186924990)

[CPU\_Top (mycpu\_top) → MMU (mmu0 和 mmu1) 10](#_Toc186924991)

[CPU\_Top (mycpu\_top) → Core (mycpu\_core) 10](#_Toc186924992)

[Core (mycpu\_core) → 流水线阶段（IF、ID、EX、MEM、WB） 11](#_Toc186924993)

[IF → ID： 11](#_Toc186924994)

[ID → EX： 11](#_Toc186924995)

[EX → MEM： 11](#_Toc186924996)

[MEM → WB： 11](#_Toc186924997)

[MEM → RF： 11](#_Toc186924998)

[WB → RF： 12](#_Toc186924999)

[Core (mycpu\_core) → 控制模块 (CTRL) 12](#_Toc186925000)

[控制模块 (CTRL) → 各流水线阶段（IF、ID、EX、MEM、WB） 12](#_Toc186925001)

[Core (mycpu\_core) → CPU\_Top (mycpu\_top) 12](#_Toc186925002)

[MMU (mmu0 和 mmu1) → 存储器接口 12](#_Toc186925003)

[二、 流水线相应阶段说明 13](#_Toc186925004)

[1.IF模块 13](#_Toc186925005)

[1.1整体说明 14](#_Toc186925006)

[1.2功能说明 14](#_Toc186925007)

[1.2.1输入信号与默认赋值 14](#_Toc186925008)

[1.2.2 程序计数器（PC）的更新 14](#_Toc186925009)

[1.2.3指令存储器使能信号的管理 14](#_Toc186925010)

[1.2.4. 下一个程序计数器（next\_pc）的计算 15](#_Toc186925011)

[1.2.5. 指令存储器接口信号的分配 15](#_Toc186925012)

[1.2.6 数据总线的打包与传递 15](#_Toc186925013)

[2.ID模块 16](#_Toc186925014)

[2.1整体说明 18](#_Toc186925015)

[2.2功能说明 18](#_Toc186925016)

[2.2.1引入定义文件和模块说明 18](#_Toc186925017)

[2.2.2. 输入信号与默认赋值 18](#_Toc186925018)

[2.2.3. 内部寄存器与信号 19](#_Toc186925019)

[2.2.4. 数据相关性处理 20](#_Toc186925020)

[2.2.5. 指令译码 20](#_Toc186925021)

[2.2.6. ALU操作码生成 20](#_Toc186925022)

[2.2.7. 访存指令处理 20](#_Toc186925023)

[2.2.8. 分支跳转处理 21](#_Toc186925024)

[2.2.9. 流水线控制 21](#_Toc186925025)

[2.2.10. 数据总线打包 21](#_Toc186925026)

[2.2.11. 分支跳转信号打包 21](#_Toc186925027)

[2.2.12. 数据相关性停顿请求 21](#_Toc186925028)

[3.EX模块 22](#_Toc186925029)

[3.1整体说明 25](#_Toc186925030)

[3.2详细功能说明 25](#_Toc186925031)

[3.2.1. 暂存数据总线与信号选择 25](#_Toc186925032)

[3.2.2. ALU 操作 25](#_Toc186925033)

[3.2.3. 乘法与除法运算处理 26](#_Toc186925034)

[3.2.4. HI/LO 寄存器处理 27](#_Toc186925035)

[3.2.5. Load 和 Store 指令处理 27](#_Toc186925036)

[3.2.6. 分支与跳转处理 28](#_Toc186925037)

[3.2.7. 流水线控制 28](#_Toc186925038)

[3.2.8. 数据总线打包与传递 28](#_Toc186925039)

[4.MEM模块 30](#_Toc186925040)

[4.1整体说明 32](#_Toc186925041)

[4.2详细功能说明 32](#_Toc186925042)

[4.2.1. 输入信号与模块说明 32](#_Toc186925043)

[4.2.2. 内部寄存器与信号 33](#_Toc186925044)

[4.2.3. 时序逻辑 33](#_Toc186925045)

[4.2.4. 访存数据处理 33](#_Toc186925046)

[4.2.5. Load 和 Store 指令处理 34](#_Toc186925047)

[4.2.6. Load-Use 数据冒险处理 35](#_Toc186925048)

[4.2.7. 数据总线打包与传递 35](#_Toc186925049)

[5.WB模块 37](#_Toc186925050)

[5.1整体说明 38](#_Toc186925051)

[5.2详细功能说明 38](#_Toc186925052)

[5.2.1. 输入信号与模块说明 38](#_Toc186925053)

[5.2.2. 内部寄存器与信号 38](#_Toc186925054)

[5.2.3. 时序逻辑 39](#_Toc186925055)

[5.2.4. 寄存器文件写回处理 39](#_Toc186925056)

[5.2.5. 数据总线打包与传递 39](#_Toc186925057)

[5.2.6. 数据选择与扩展逻辑 40](#_Toc186925058)

[5.2.7. 寄存器文件写回数据处理 40](#_Toc186925059)

[6.CTRL模块 41](#_Toc186925060)

[6.1整体说明 42](#_Toc186925061)

[6.2详细功能说明 42](#_Toc186925062)

[6.2.1. 输入信号与模块说明 42](#_Toc186925063)

[6.2.2. 内部逻辑与停顿信号生成 42](#_Toc186925064)

[6.2.3. 停顿信号总线 stall 的具体设置 43](#_Toc186925065)

[7.HI\_LO\_REG模块 45](#_Toc186925066)

[7.1整体说明 46](#_Toc186925067)

[7.2详细功能说明 46](#_Toc186925068)

[7.2.1. 输入信号与模块说明 46](#_Toc186925069)

[7.2.2. 内部寄存器与数据存储 47](#_Toc186925070)

[7.2.3. HI/LO 寄存器的读出 47](#_Toc186925071)

[总结与思考 48](#_Toc186925072)

# 分工及总体框架

## 1.分工

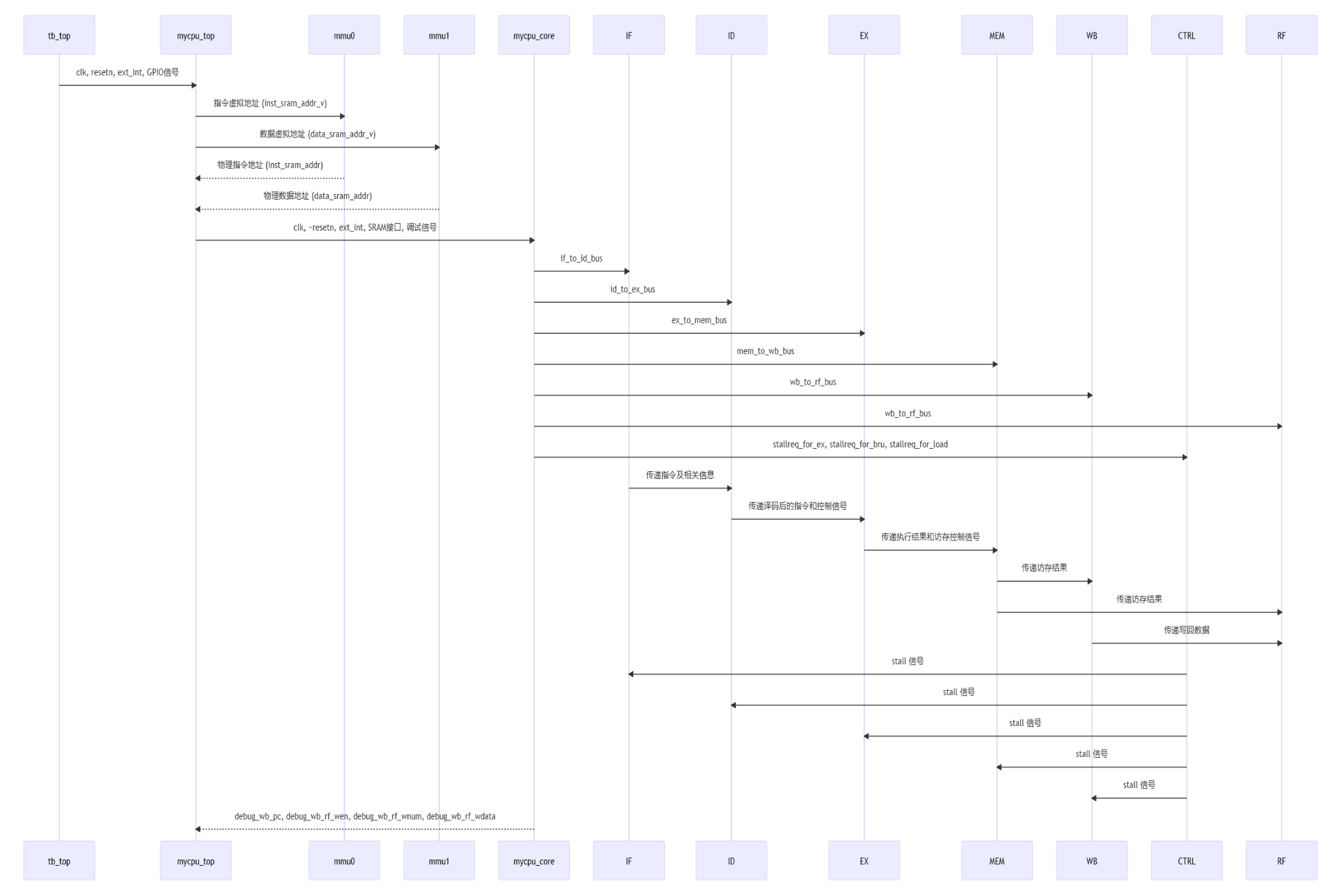
|  |  |  |
| --- | --- | --- |
| 姓名 | 分工 | 工作量占比 |
| 刘庆然 | 整体设计以及报告编写 | 65% |
| 马智鸿 | 参与部分实现、参与报告编写 | 35% |

## 2.运行环境及所用工具

由于提供的CG环境较卡，并且保存记录、查看输出调试等不太方便，最终在本地使用 vivado HLS 2019.2在本地进行仿真运行。

使用github创建仓库进行管理，并使用gitkraken、github desktop等可视化工具来拉取、上传代码等。

## 3.整体框架



### 1. 模块功能概述：

#### Testbench (tb\_top)

功能：顶层测试模块，负责配置和驱动仿真环境，实例化被测的 CPU 顶层模块 (mycpu\_top) 以及其他必要的组件。

连接：通过时钟 (clk)、复位信号 (resetn)、外部中断信号 (ext\_int) 和 GPIO 信号连接到 CPU 顶层模块。

#### CPU\_Top (mycpu\_top)

功能：CPU 的顶层模块，整合了核心处理器模块 (mycpu\_core) 和内存管理单元（MMU）模块 (mmu0 和 mmu1)，提供与外部指令存储器和数据存储器的接口，以及管理中断信号和调试信号。

**连接：**

接收来自测试平台的时钟、复位、中断和 GPIO 信号。

将指令虚拟地址 (inst\_sram\_addr\_v) 和数据虚拟地址 (data\_sram\_addr\_v) 传递给 MMU 模块 (mmu0 和 mmu1) 进行地址转换。

接收 MMU 模块返回的物理地址 (inst\_sram\_addr 和 data\_sram\_addr)。

连接到核心处理器模块，传递时钟、复位、中断、SRAM 接口和调试信号。

#### MMU (mmu0 和 mmu1)

功能：内存管理单元，负责将虚拟地址转换为物理地址，确保指令和数据访问的正确性和安全性。

连接：

mmu0：接收指令虚拟地址 (inst\_sram\_addr\_v)，输出物理指令地址 (inst\_sram\_addr) 至指令存储器。

mmu1：接收数据虚拟地址 (data\_sram\_addr\_v)，输出物理数据地址 (data\_sram\_addr) 至数据存储器。

#### Core (mycpu\_core)

功能：核心处理器模块，负责实现 CPU 的五级流水线（IF、ID、EX、MEM、WB）以及控制模块 (CTRL)。

连接：

实例化并连接五个流水线阶段模块（IF、ID、EX、MEM、WB）。

发送停顿请求信号（stallreq\_for\_ex、stallreq\_for\_bru、stallreq\_for\_load）到控制模块 (CTRL)。

将写回阶段的调试信号（debug\_wb\_pc、debug\_wb\_rf\_wen、debug\_wb\_rf\_wnum、debug\_wb\_rf\_wdata）传递回 CPU 顶层模块 (mycpu\_top)。

#### 流水线阶段子模块（IF、ID、EX、MEM、WB）

功能：

IF（指令取出）：负责从指令存储器中取出指令，并将取出的指令及相关信息传递给 ID 阶段。

ID（指令译码）：负责对取出的指令进行译码，读取寄存器文件中的操作数，并将译码后的指令和控制信号传递给 EX 阶段。

EX（执行）：负责执行指令，包括算术逻辑运算、地址计算等，并将执行结果传递给 MEM 阶段。

MEM（访存）：负责执行访存操作（如 Load 和 Store 指令），并将访存结果传递给 WB 阶段。

WB（写回）：负责将执行或访存的结果写回寄存器文件，并提供调试信号。

#### 控制模块 (CTRL)

功能：管理 CPU 流水线的停顿信号。根据来自不同流水线阶段的停顿请求信号（如执行阶段、分支操作、Load 操作），生成整体的停顿信号总线 (stall)，控制各个流水线阶段的停顿与恢复。

连接：

接收来自核心处理器模块的停顿请求信号（stallreq\_for\_ex、stallreq\_for\_bru、stallreq\_for\_load）。

向所有流水线阶段（IF、ID、EX、MEM、WB）输出停顿信号 (stall)。

#### RF（寄存器文件）

功能：存储和提供 CPU 寄存器的数据，供指令译码和执行阶段使用。

连接：

接收来自 MEM 和 WB 阶段的写回数据。

### 2. 模块之间的连接关系及其功能：

#### Testbench (tb\_top) → CPU\_Top (mycpu\_top)

连接信号：clk（时钟信号）、resetn（复位信号）、ext\_int（外部中断信号）、GPIO 信号（如 LED、开关、按键等）。

功能：测试平台驱动 CPU 顶层模块，提供必要的输入信号。

#### CPU\_Top (mycpu\_top) → MMU (mmu0 和 mmu1)

连接信号：

mmu0：接收指令虚拟地址 (inst\_sram\_addr\_v)，输出物理指令地址 (inst\_sram\_addr) 至指令存储器。

mmu1：接收数据虚拟地址 (data\_sram\_addr\_v)，输出物理数据地址 (data\_sram\_addr) 至数据存储器。

功能：CPU 顶层模块通过 MMU 模块将虚拟地址转换为物理地址，确保指令和数据的正确访问。

#### CPU\_Top (mycpu\_top) → Core (mycpu\_core)

连接信号：clk（时钟信号）、~resetn（复位信号，取反后高电平有效）、ext\_int（中断信号）、SRAM 接口信号（指令和数据存储器的使能、写使能、地址、写数据、读数据）、调试信号。

功能：CPU 顶层模块实例化核心处理器模块，并传递必要的控制和数据信号。

#### Core (mycpu\_core) → 流水线阶段（IF、ID、EX、MEM、WB）

连接信号：各阶段的数据总线（如 if\_to\_id\_bus、id\_to\_ex\_bus、ex\_to\_mem\_bus、mem\_to\_wb\_bus）、寄存器文件的数据总线（如 wb\_to\_rf\_bus、mem\_to\_rf\_bus）。

功能：核心处理器模块实例化并连接五个流水线阶段模块，传递指令和数据。

流水线阶段之间的数据传递

##### IF → ID：

连接信号：if\_to\_id\_bus（指令取出到译码的数据总线）。

功能：传递取出的指令及相关信息至译码阶段。

##### ID → EX：

连接信号：id\_to\_ex\_bus（译码到执行的数据总线）。

功能：传递译码后的指令和控制信号至执行阶段。

##### EX → MEM：

连接信号：ex\_to\_mem\_bus（执行到访存的数据总线）。

功能：传递执行结果和访存控制信号至访存阶段。

##### MEM → WB：

连接信号：mem\_to\_wb\_bus（访存到写回的数据总线）。

功能：传递访存结果至写回阶段。

##### MEM → RF：

连接信号：mem\_to\_rf\_bus（访存到寄存器文件的数据总线）。

功能：将访存结果写入寄存器文件。

##### WB → RF：

连接信号：wb\_to\_rf\_bus（写回到寄存器文件的数据总线）。

功能：将写回结果写入寄存器文件。

#### Core (mycpu\_core) → 控制模块 (CTRL)

连接信号：stallreq\_for\_ex、stallreq\_for\_bru、stallreq\_for\_load（来自执行、分支和 Load 操作的停顿请求信号）。

功能：将停顿请求信号传递给控制模块，以生成整体的停顿信号。

#### 控制模块 (CTRL) → 各流水线阶段（IF、ID、EX、MEM、WB）

连接信号：stall（停顿信号总线）。

功能：控制各流水线阶段的停顿与恢复，确保流水线在数据相关或控制相关情况下正确运行。

#### Core (mycpu\_core) → CPU\_Top (mycpu\_top)

连接信号：debug\_wb\_pc、debug\_wb\_rf\_wen、debug\_wb\_rf\_wnum、debug\_wb\_rf\_wdata（写回阶段的调试信号）。

功能：将写回阶段的调试信息传递至 CPU 顶层模块，便于测试和监控。

#### MMU (mmu0 和 mmu1) → 存储器接口

mmu0 → inst\_sram\_addr：指令存储器的物理地址输入。

mmu1 → data\_sram\_addr：数据存储器的物理地址输入。

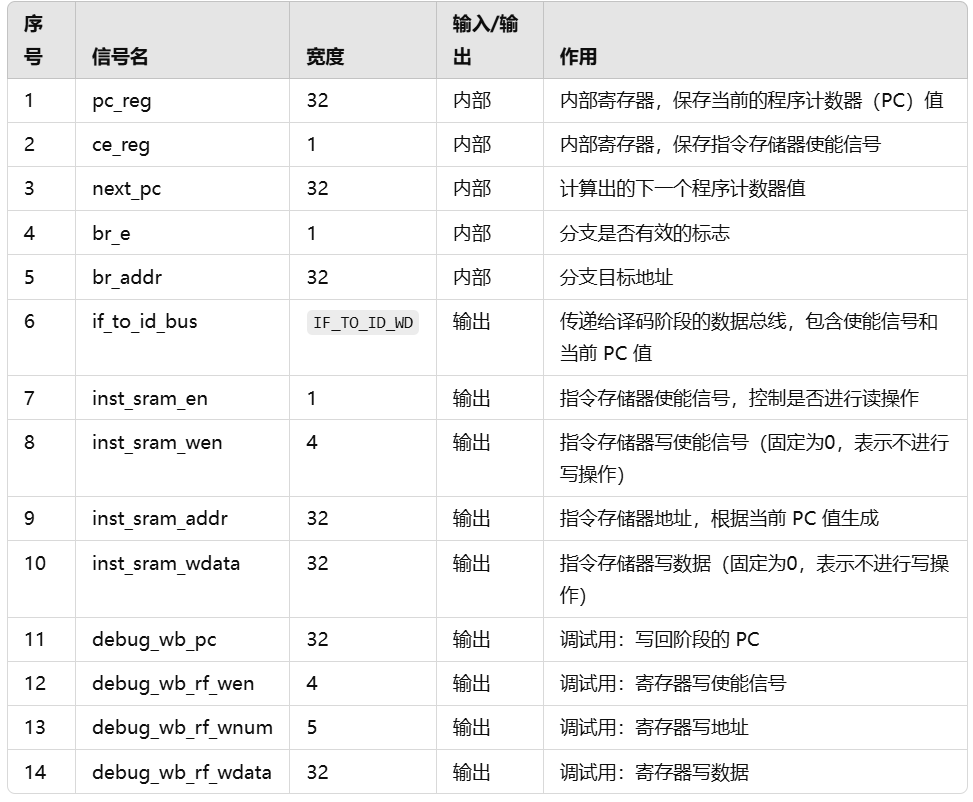
功能：将转换后的物理地址传递给对应的存储器接口，确保指令和数据的正确访问。

# 流水线相应阶段说明

## 1.IF模块

IF模块输入输出：

IF 模块详细功能说明



### 1.1整体说明

IF（Instruction Fetch，指令取出）模块负责从指令存储器中取出指令，并将取出的指令及相关信息传递给 ID（Instruction Decode，指令译码）阶段。该模块管理程序计数器（PC），处理分支跳转，并根据停顿信号控制流水线的暂停与恢复。

### 1.2功能说明

#### 1.2.1输入信号与默认赋值

时钟信号 clk 和复位信号 rst：

clk：同步时钟信号，驱动模块内部的时序逻辑。

rst：复位信号，高电平有效，用于初始化程序计数器和使能信号。

停顿信号 stall：

用于控制流水线的暂停。当 stall[0] 为 NoStop 时，允许 PC 更新；否则，保持当前 PC 值。

分支信息总线 br\_bus：

包含分支有效标志 br\_e 和分支目标地址 br\_addr。用于决定是否跳转到新的 PC 地址。

#### 1.2.2 程序计数器（PC）的更新

复位逻辑：

当 rst 为高电平时，将 pc\_reg 初始化为特定地址 32'hbfbf\_fffc，表示程序的起始位置。

PC 更新逻辑：

在每个时钟上升沿，如果没有停顿信号 (stall[0] == NoStop)，则更新 pc\_reg 为 next\_pc。

如果有停顿信号，则保持当前的 pc\_reg 值，暂停指令取出。

#### 1.2.3指令存储器使能信号的管理

复位逻辑：

当 rst 为高电平时，将指令存储器使能信号 ce\_reg 置为低电平，禁用指令存储器。

使能信号更新逻辑：

在每个时钟上升沿，如果没有停顿信号，则将 ce\_reg 置为高电平，启用指令存储器。

如果有停顿信号，则保持当前的 ce\_reg 状态（通常为高电平）。

#### 1.2.4. 下一个程序计数器（next\_pc）的计算

分支跳转处理：

如果分支有效标志 br\_e 为高电平，则将 br\_addr 作为下一个 PC 值，实现跳转。

否则，将当前 PC 值加 4，顺序执行下一条指令。

#### 1.2.5. 指令存储器接口信号的分配

使能信号 inst\_sram\_en：

由 ce\_reg 决定，控制是否进行指令存取操作。

写使能信号 inst\_sram\_wen：

固定为 4'b0，表示不进行写操作。

指令存储器地址 inst\_sram\_addr：

由当前 PC 值 pc\_reg 决定，指向要取出的指令地址。

指令存储器写数据 inst\_sram\_wdata：

固定为 32'b0，表示没有写入数据。

#### 1.2.6 数据总线的打包与传递

if\_to\_id\_bus：

将指令存储器使能信号 ce\_reg 和当前 PC 值 pc\_reg 打包成总线，传递给 ID 阶段，包含：

高位：ce\_reg（指令存储器使能信号）

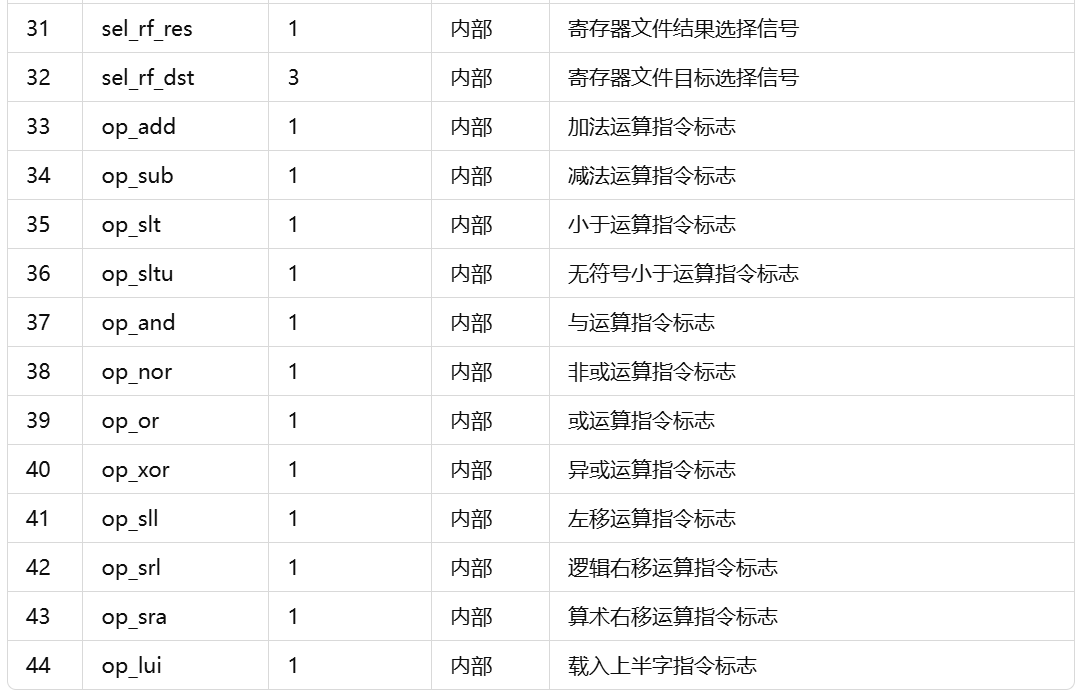
低位：pc\_reg（当前程序计数器值）

## 2.ID模块

ID模块输入输出



ID模块详细功能说明：



### 2.1整体说明

ID 模块负责将从 IF（Instruction Fetch，指令取出）阶段传递过来的指令进行译码，读取所需的寄存器数据，处理数据相关性，并生成传递给 EX（Execution，执行）阶段的控制信号。该模块还处理分支跳转逻辑，并与寄存器文件和其他流水线阶段交互，确保指令正确执行。

### 2.2功能说明

#### 2.2.1引入定义文件和模块说明

include "lib/defines.vh"：

引入包含各种宏定义和参数的头文件，便于在代码中统一使用这些定义。

模块注释：

说明了 ID 模块的基本功能，包括指令解码、寄存器读取、数据相关处理和控制信号生成。

#### 2.2.2. 输入信号与默认赋值

时钟信号 clk 和复位信号 rst：

clk：同步时钟信号，驱动模块内部的时序逻辑。

rst：复位信号，高电平有效，用于初始化模块。

停顿信号 stall：

用于控制流水线的暂停。当 stall[1] 为 Stop 时，控制该阶段的暂停。

数据总线和信号：

if\_to\_id\_bus：从 IF 阶段传递过来的数据总线，包含使能信号和当前 PC 值。

inst\_sram\_rdata：从指令存储器读取的指令数据。

ex\_id：来自执行阶段的标识信号，用于数据转发。

写回阶段数据总线：

wb\_to\_rf\_bus：从写回阶段到寄存器文件的数据总线。

ex\_to\_rf\_bus：从执行阶段到寄存器文件的数据总线。

mem\_to\_rf\_bus：从访存阶段到寄存器文件的数据总线。

HI/LO寄存器信号：

ex\_hi\_lo\_bus：来自执行阶段的 HI/LO 寄存器相关信号。

id\_hi\_lo\_bus：传递给执行阶段的 HI/LO 寄存器相关信号。

#### 2.2.3. 内部寄存器与信号

寄存器：

if\_to\_id\_bus\_r：暂存从 IF 阶段传递过来的数据总线。

pc\_reg、ce\_reg：程序计数器和使能信号寄存器。

flag、buf\_inst：用于处理停顿期间的指令缓存和状态标志。

寄存器文件：

实例化 regfile 模块，读取源操作数 rs 和 rt 的数据。

HI/LO寄存器：

实例化 hi\_lo\_reg 模块，处理 HI/LO 寄存器的读写操作。

#### 2.2.4. 数据相关性处理

数据转发：

通过比较 rs 和 rt 寄存器地址与写回阶段的写地址，选择最新的数据写回寄存器，避免数据相关性带来的问题。

寄存器文件写使能信号：

根据不同指令类型决定是否写回寄存器。

#### 2.2.5. 指令译码

指令字段解码：

从指令中提取操作码、寄存器字段、功能码、立即数等信息。

指令类型判定：

通过解码操作码和功能码，确定指令类型（算术运算、逻辑运算、移位、分支跳转、数据移动、访存等）。

#### 2.2.6. ALU操作码生成

算术运算指令：

对加、减、小于等指令进行组合，生成相应的 ALU 操作码。

逻辑运算指令：

对与、或、异或等指令进行组合，生成相应的 ALU 操作码。

移位指令：

对左移、右移等指令进行组合，生成相应的 ALU 操作码。

其他指令：

对特殊指令如 lui 进行处理。

#### 2.2.7. 访存指令处理

Load和Store使能信号：

根据指令类型，设置数据存储器的使能和写使能信号。

#### 2.2.8. 分支跳转处理

分支条件判断：

根据寄存器数据和指令类型，判断是否满足跳转条件，生成分支信号 br\_e 和跳转地址 br\_addr。

#### 2.2.9. 流水线控制

停顿请求信号：

当执行阶段需要转发数据时，生成停顿请求信号 stallreq\_for\_bru，通知流水线暂停，防止数据相关性错误。

#### 2.2.10. 数据总线打包

id\_to\_ex\_bus：

将译码结果和控制信号打包成总线，传递给执行阶段（EX），包括当前 PC 值、指令、ALU 操作码、数据源选择信号、访存信号、寄存器写使能和地址、处理后的操作数等。

id\_hi\_lo\_bus、id\_load\_bus、id\_save\_bus：

将 HI/LO 寄存器相关信号和访存指令信号打包传递给执行阶段。

#### 2.2.11. 分支跳转信号打包

br\_bus：

将分支条件满足标志 br\_e 和分支跳转地址 br\_addr 打包传递给下游模块。

#### 2.2.12. 数据相关性停顿请求

stallreq\_for\_bru：

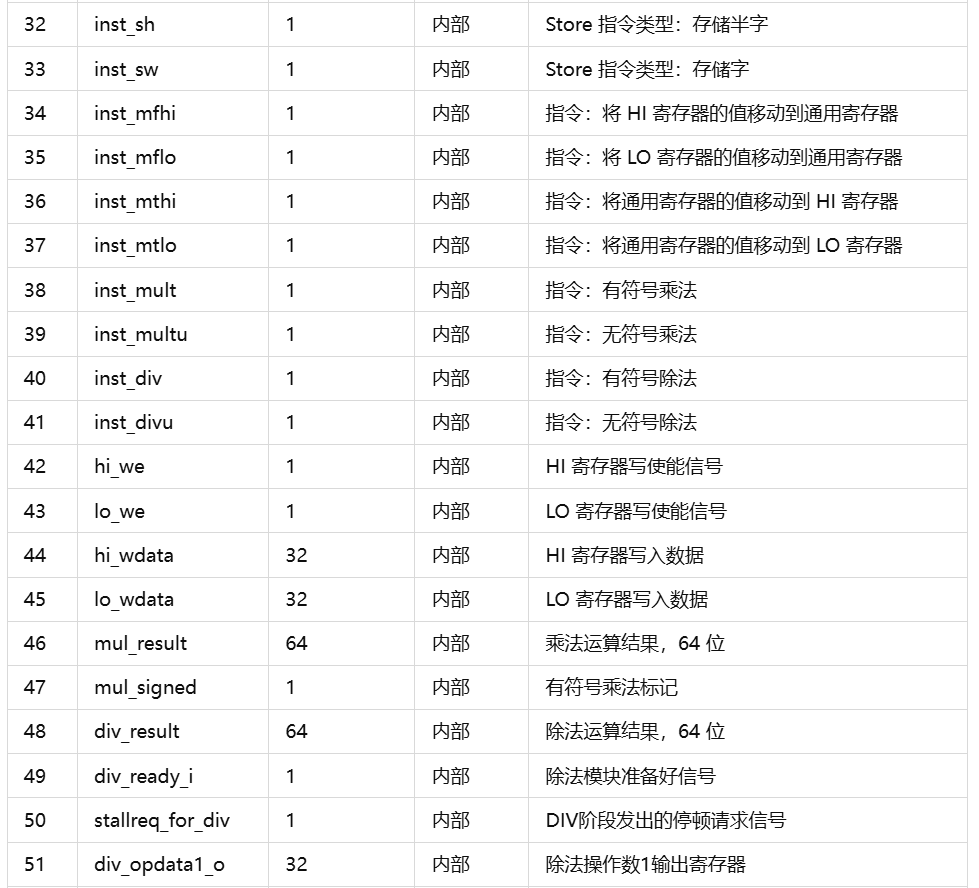
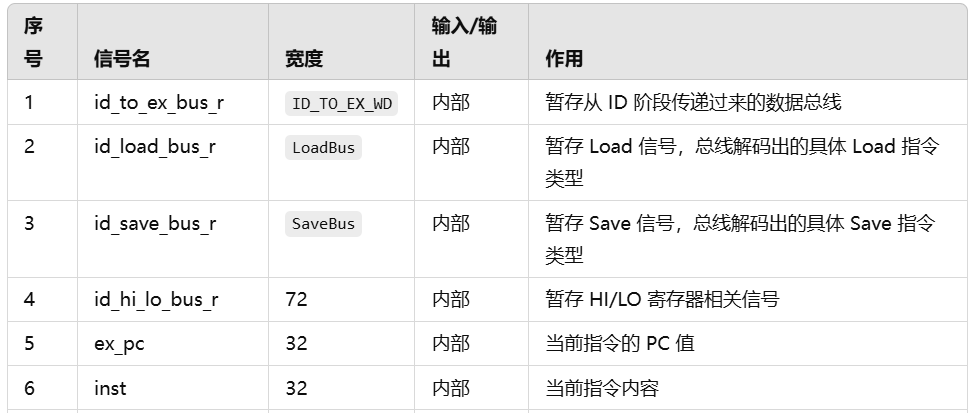
当执行阶段需要转发的数据与当前 ID 阶段的源寄存器相同时，发出停顿请求信号，暂停流水线，确保数据的正确性。

## 3.EX模块

EX模块输入输出：



EX模块详细功能说明



### 3.1整体说明

EX 模块负责在流水线的执行阶段执行算术和逻辑运算，计算访存地址，处理数据转发，以及管理特殊寄存器（如 HI/LO）的读写操作。该模块通过与 ALU、乘法器、除法器等子模块协作，实现指令的执行，并将结果传递到下一个流水线阶段（MEM）。此外，EX 模块还负责处理数据相关性，确保流水线的正确性和效率。

### 3.2详细功能说明

#### 3.2.1. 暂存数据总线与信号选择

寄存器 id\_to\_ex\_bus\_r、id\_load\_bus\_r、id\_save\_bus\_r 和 id\_hi\_lo\_bus\_r：

这些寄存器用于暂存从 ID 阶段传递过来的数据总线、Load/Save 信号以及 HI/LO 寄存器相关信号，确保在流水线停顿期间指令数据的稳定性。

时钟上升沿逻辑：

复位 (rst 为高电平)：

清空所有暂存寄存器，恢复初始状态。

停顿信号处理：

如果 stall[2] 为 Stop 且 stall[3] 为 NoStop，则清空寄存器。

如果 stall[2] 为 NoStop，则更新寄存器，将 ID 阶段的数据传递到 EX 阶段。

#### 3.2.2. ALU 操作

操作数选择：

ALU 源操作数1 (alu\_src1)：

如果 sel\_alu\_src1[1] 为 1，选择当前指令的 PC (ex\_pc) 作为操作数1。

如果 sel\_alu\_src1[2] 为 1，选择位移量扩展后的值 (sa\_zero\_extend) 作为操作数1。

否则，选择寄存器读取的数据1 (rf\_rdata1) 作为操作数1。

ALU 源操作数2 (alu\_src2)：

如果 sel\_alu\_src2[1] 为 1，选择符号扩展的立即数 (imm\_sign\_extend) 作为操作数2。

如果 sel\_alu\_src2[2] 为 1，选择常数8作为操作数2。

如果 sel\_alu\_src2[3] 为 1，选择零扩展的立即数 (imm\_zero\_extend) 作为操作数2。

否则，选择寄存器读取的数据2 (rf\_rdata2) 作为操作数2。

ALU 实例化：

alu 模块根据 alu\_op、alu\_src1 和 alu\_src2 执行相应的算术或逻辑运算，输出运算结果 alu\_result。

EX 阶段结果选择 (ex\_result)：

如果当前指令是 MFHI，则选择 HI 寄存器的值作为 EX 阶段的结果。

如果当前指令是 MFLO，则选择 LO 寄存器的值作为 EX 阶段的结果。

否则，选择 ALU 的运算结果 alu\_result 作为 EX 阶段的结果。

#### 3.2.3. 乘法与除法运算处理

乘法运算：

有符号乘法标记 (mul\_signed)：

如果当前指令是有符号乘法指令 MULT，则 mul\_signed 为 1。

否则，mul\_signed 为 0。

乘法器实例化：

mul 模块根据 mul\_signed 执行有符号或无符号乘法运算，输出 64 位的结果 mul\_result。

除法运算：

除法模块实例化：

div 模块根据 signed\_div\_o 执行有符号或无符号除法运算，输出 64 位的结果 div\_result 和准备好信号 div\_ready\_i。

除法控制逻辑：

当当前指令是除法指令（DIV 或 DIVU）且除法结果未准备好时，启动除法运算并发出停顿请求 stallreq\_for\_div，暂停流水线。

当除法结果准备好时，停止除法运算并解除停顿请求。

#### 3.2.4. HI/LO 寄存器处理

写使能信号：

hi\_we：

如果当前指令是 MTHI、MULT、MULTU、DIV 或 DIVU，则 HI 寄存器写使能信号 hi\_we 为 1。

lo\_we：

如果当前指令是 MTLO、MULT、MULTU、DIV 或 DIVU，则 LO 寄存器写使能信号 lo\_we 为 1。

写入数据选择：

HI 寄存器写入数据 (hi\_wdata)：

如果当前指令是 MTHI，则将寄存器读取的数据1 (rf\_rdata1) 写入 HI 寄存器。

如果当前指令是 MULT 或 MULTU，则将乘法结果的高 32 位 (mul\_result[63:32]) 写入 HI 寄存器。

如果当前指令是 DIV 或 DIVU，则将除法结果的高 32 位 (div\_result[63:32]) 写入 HI 寄存器。

否则，写入 0。

LO 寄存器写入数据 (lo\_wdata)：

如果当前指令是 MTLO，则将寄存器读取的数据1 (rf\_rdata1) 写入 LO 寄存器。

如果当前指令是 MULT 或 MULTU，则将乘法结果的低 32 位 (mul\_result[31:0]) 写入 LO 寄存器。

如果当前指令是 DIV 或 DIVU，则将除法结果的低 32 位 (div\_result[31:0]) 写入 LO 寄存器。

否则，写入 0。

#### 3.2.5. Load 和 Store 指令处理

Load 和 Store 指令类型解码：

从 id\_load\_bus\_r 和 id\_save\_bus\_r 中解码出具体的 Load 和 Store 指令类型，如 LB、LBU、LH、LHU、LW、SB、SH、SW。

Load 信号传递：

将解码出的 Load 指令类型信号打包传递给下一个阶段，用于访存操作。

Store 信号传递：

根据 Store 指令类型和 ALU 计算出的地址，生成字节选择信号 data\_ram\_sel，以确定哪些字节需要写入数据存储器。

数据RAM 接口信号生成：

data\_ram\_en：由 Load 或 Store 指令决定，控制数据存储器的使能。

data\_ram\_wen：根据 Store 指令类型和字节选择信号，控制写入的数据字节。

data\_sram\_addr：由 ALU 计算出的地址决定。

data\_sram\_wdata：根据 Store 指令类型，选择要写入的数据：

SB：将寄存器读取的数据的最低 8 位扩展并重复到所有字节。

SH：将寄存器读取的数据的最低 16 位扩展并重复到两个字节。

SW 和其他 Store 指令：直接写入 32 位数据。

#### 3.2.6. 分支与跳转处理

分支跳转信号打包：

将分支条件满足标志 br\_e 和跳转地址 br\_addr 打包成 br\_bus，传递给下游模块，用于更新程序计数器。

#### 3.2.7. 流水线控制

停顿请求信号 stallreq\_for\_ex：

当除法运算未完成时，发出停顿请求信号 stallreq\_for\_div，并将其传递给 EX 阶段的停顿请求信号 stallreq\_for\_ex，暂停流水线，直到除法结果准备好。

#### 3.2.8. 数据总线打包与传递

ex\_to\_mem\_bus：

将 EX 阶段的结果和控制信号打包成总线，传递给 MEM 阶段，包括当前 PC、数据RAM 使能和写使能信号、寄存器文件写使能和写地址，以及 EX 阶段的运算结果。

ex\_to\_rf\_bus：

将 EX 阶段的结果打包成总线，传递给寄存器文件，用于寄存器写回。

ex\_hi\_lo\_bus：

将 EX 阶段的 HI/LO 寄存器相关信号打包传递给下一阶段，用于管理特殊寄存器的读写操作。

ex\_load\_bus：

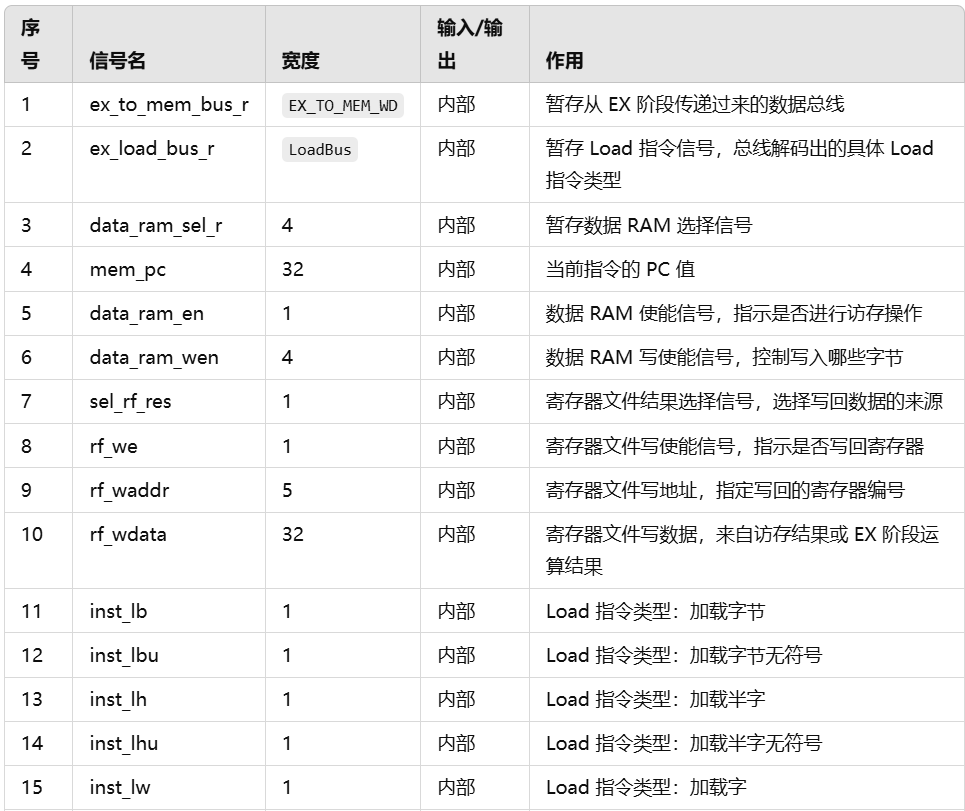
将 EX 阶段的 Load 指令类型信号打包传递给下一个阶段，用于访存操作。

## 4.MEM模块

MEM 模块输入输出



MEM 模块详细功能说明



### 4.1整体说明

MEM 模块负责在流水线的访存阶段处理与内存相关的操作，包括加载（Load）和存储（Store）指令。它从执行阶段（EX）接收运算结果和控制信号，根据指令类型访问数据存储器（Data SRAM），读取或写入数据，并将结果传递到写回阶段（WB）。此外，MEM 模块还处理数据相关性，通过生成停顿请求信号（stallreq\_for\_load）来避免 Load-Use 数据冒险。

### 4.2详细功能说明

#### 4.2.1. 输入信号与模块说明

时钟信号 clk 和复位信号 rst：

clk：同步时钟信号，驱动模块内部的时序逻辑。

rst：复位信号，高电平有效，用于初始化模块内部寄存器和状态。

停顿信号 stall：

用于控制流水线的暂停。当 stall[3] 为 Stop 时，控制该阶段的暂停。

数据总线与信号：

ex\_to\_mem\_bus：从 EX 阶段传递到 MEM 阶段的数据总线，包含当前指令的 PC、指令内容、ALU 操作码、操作数选择信号、访存信号、寄存器写使能和写地址等信息。

data\_sram\_rdata：从数据 SRAM 读取的数据，用于 Load 指令。

data\_ram\_sel：数据 RAM 的字节选择信号，根据 Store 指令类型和地址确定写入的字节。

ex\_load\_bus：从 EX 阶段传递的 Load 指令信号，指示当前指令的 Load 类型（如 LB、LBU、LH、LHU、LW）。

输出数据总线：

mem\_to\_wb\_bus：将访存阶段的结果和控制信号打包传递到写回阶段（WB）。

mem\_to\_rf\_bus：将访存阶段的结果传递给寄存器文件，用于寄存器写回。

stallreq\_for\_load：对 Load 操作发出的停顿请求信号，用于处理 Load-Use 数据冒险。

#### 4.2.2. 内部寄存器与信号

寄存器：

ex\_to\_mem\_bus\_r：暂存从 EX 阶段传递过来的数据总线，确保在停顿期间指令数据的稳定性。

ex\_load\_bus\_r：暂存 Load 指令信号。

data\_ram\_sel\_r：暂存数据 RAM 选择信号。

信号解码：

从 ex\_to\_mem\_bus\_r 中解码出当前指令的 PC (mem\_pc)、指令内容、ALU 操作码、操作数选择信号、访存信号、寄存器写使能和写地址，以及寄存器读取的数据。

从 ex\_load\_bus\_r 中解码出具体的 Load 指令类型信号，如 LB、LBU、LH、LHU、LW。

#### 4.2.3. 时序逻辑

寄存器更新：

在每个时钟上升沿，根据信号 rst 和 stall 的状态更新寄存器：

复位：清空 ex\_to\_mem\_bus\_r、ex\_load\_bus\_r 和 data\_ram\_sel\_r。

停顿信号处理：

如果 stall[3] 为 Stop 且 stall[4] 为 NoStop，则清空寄存器。

如果 stall[3] 为 NoStop，则更新寄存器，将 EX 阶段的数据传递到 MEM 阶段。

流水线停顿：

如果 stall[3] 为 Stop 且 stall[4] 也是 Stop，则保持当前寄存器值，暂停流水线。

#### 4.2.4. 访存数据处理

数据选择与扩展：

字节数据 (b\_data)：

根据 data\_ram\_sel\_r[3:0] 的值，选择数据 SRAM 读取的数据的特定字节。

半字数据 (h\_data)：

根据 data\_ram\_sel\_r 的值，选择数据 SRAM 读取的特定半字（16 位）。

字数据 (w\_data)：

直接使用数据 SRAM 读取的 32 位数据。

访存结果选择：

根据 Load 指令类型，选择并扩展数据：

LB（Load Byte）：符号扩展字节数据。

LBU（Load Byte Unsigned）：零扩展字节数据。

LH（Load Halfword）：符号扩展半字数据。

LHU（Load Halfword Unsigned）：零扩展半字数据。

LW（Load Word）：直接使用字数据。

其他情况，结果为 0。

寄存器文件写回数据选择：

如果 sel\_rf\_res 和 data\_ram\_en 为 1，则将访存结果 mem\_result 写回寄存器文件。

否则，使用来自 EX 阶段的运算结果 ex\_result。

#### 4.2.5. Load 和 Store 指令处理

Load 指令类型解码：

从 ex\_load\_bus\_r 中解码出具体的 Load 指令类型信号，如 LB、LBU、LH、LHU、LW。

Store 指令类型解码：

从 data\_ram\_sel\_r 中解码出具体的 Store 指令类型信号，如 SB、SH、SW。

数据 RAM 选择信号生成：

根据 Store 指令类型和 ALU 计算出的地址，生成字节选择信号 data\_ram\_sel：

SB（Store Byte）、LB、LBU：选择特定的一个字节。

SH（Store Halfword）、LH、LHU：选择特定的两个字节。

SW（Store Word）、LW：选择全部四个字节。

数据 SRAM 接口信号生成：

data\_sram\_en：由 Load 或 Store 指令决定，控制数据 SRAM 的使能。

data\_sram\_wen：根据 Store 指令类型和字节选择信号，控制写入的数据字节：

使用 data\_ram\_wen 和 data\_ram\_sel\_r 的值，通过按位与操作确定写使能。

data\_sram\_addr：由 ALU 计算出的地址决定。

data\_sram\_wdata：

SB（Store Byte）：将寄存器读取的数据的最低 8 位扩展并重复到所有字节。

SH（Store Halfword）：将寄存器读取的数据的最低 16 位扩展并重复到两个字节。

SW（Store Word）：直接写入 32 位数据。

#### 4.2.6. Load-Use 数据冒险处理

停顿请求信号 stallreq\_for\_load：

待实现：根据设计需求，需要检测当前阶段的 Load 指令是否与后续指令存在数据相关性（即 Load 指令加载的数据被紧随其后的指令使用）。如果存在数据相关性，发出停顿请求信号 stallreq\_for\_load，暂停流水线，直到数据加载完成并可用于后续指令。

实现建议：

Load-Use 检测：

检测当前 Load 指令的目标寄存器是否被下一指令使用。

如果是，则需要暂停流水线，直到 Load 指令完成数据加载。

停顿信号生成：

如果检测到 Load-Use 数据冒险，设置 stallreq\_for\_load 为 1，否则为 0。

#### 4.2.7. 数据总线打包与传递

mem\_to\_wb\_bus：

将访存阶段的结果和控制信号打包传递到写回阶段（WB），包括：

当前指令的 PC (mem\_pc)。

寄存器文件写使能信号 (rf\_we)。

寄存器文件写地址 (rf\_waddr)。

寄存器文件写数据 (rf\_wdata)。

mem\_to\_rf\_bus：

将访存阶段的结果打包传递给寄存器文件，用于寄存器写回，包含：

寄存器文件写使能信号 (rf\_we)。

寄存器文件写地址 (rf\_waddr)。

寄存器文件写数据 (rf\_wdata)。

## 5.WB模块

WB 模块输入输出



WB 模块详细功能说明



### 5.1整体说明

WB 模块负责在流水线的写回阶段将处理后的结果写回到寄存器堆中。它从 MEM（Memory Access，访存）阶段接收数据和控制信号，解包这些信号后，将结果传递给寄存器文件（Register File）。此外，WB 模块还提供了一些用于调试的输出信号，以便监控写回阶段的状态。

### 5.2详细功能说明

#### 5.2.1. 输入信号与模块说明

时钟信号 clk 和复位信号 rst：

clk：同步时钟信号，驱动模块内部的时序逻辑。

rst：复位信号，高电平有效，用于初始化模块内部寄存器和状态。

停顿信号 stall：

用于控制流水线的暂停。当 stall[4] 为 Stop 时，控制该阶段的暂停。

数据总线与信号：

mem\_to\_wb\_bus：从 MEM 阶段传递到 WB 阶段的数据总线，包含当前指令的 PC、寄存器写使能信号、写地址以及写数据等信息。

data\_ram\_sel：数据 RAM 的字节选择信号，根据 Store 指令类型和地址确定写入的字节。

输出数据总线：

wb\_to\_rf\_bus：将写回阶段的结果传递给寄存器文件，用于寄存器写回。

debug\_wb\_\*：用于调试目的，输出写回阶段的 PC、寄存器写使能信号、写地址和写数据。

#### 5.2.2. 内部寄存器与信号

寄存器：

mem\_to\_wb\_bus\_r：暂存从 MEM 阶段传递过来的数据总线，确保在停顿期间指令数据的稳定性。

信号解包：

从 mem\_to\_wb\_bus\_r 中解包出当前指令的 PC (wb\_pc)、寄存器写使能信号 (rf\_we)、寄存器写地址 (rf\_waddr) 和寄存器写数据 (rf\_wdata)。

#### 5.2.3. 时序逻辑

寄存器更新：

在每个时钟上升沿，根据信号 rst 和 stall 的状态更新寄存器：

复位：清空 mem\_to\_wb\_bus\_r，恢复初始状态。

停顿信号处理：

如果 stall[4] 为 Stop 且 stall[5] 为 NoStop，则清空寄存器。

如果 stall[4] 为 NoStop，则更新寄存器，将 MEM 阶段的数据传递到 WB 阶段。

如果 stall[4] 和 stall[5] 均为 Stop，则保持当前寄存器值，暂停流水线。

#### 5.2.4. 寄存器文件写回处理

寄存器文件写回数据选择 rf\_wdata：

根据寄存器文件结果选择信号 sel\_rf\_res 和数据 RAM 使能信号 data\_ram\_en，选择写回数据的来源：

如果 sel\_rf\_res 和 data\_ram\_en 为 1，则选择访存结果 mem\_result 作为写回数据。

否则，使用来自 EX 阶段的运算结果 ex\_result 作为写回数据。

#### 5.2.5. 数据总线打包与传递

wb\_to\_rf\_bus：

将寄存器文件写使能信号 (rf\_we)、写地址 (rf\_waddr) 和写数据 (rf\_wdata) 打包成总线，传递给寄存器文件，用于寄存器写回。

调试信号输出：

debug\_wb\_pc：输出当前指令的 PC 值，便于调试。

debug\_wb\_rf\_wen：将寄存器写使能信号扩展为 4 位，便于调试。

debug\_wb\_rf\_wnum：输出寄存器写地址，便于调试。

debug\_wb\_rf\_wdata：输出寄存器写数据，便于调试。

#### 5.2.6. 数据选择与扩展逻辑

字节数据选择 (b\_data)：

根据 data\_ram\_sel\_r 的值，选择数据 SRAM 读取的数据的特定字节：

data\_ram\_sel\_r[3] 为 1：选择最高 8 位字节 (31:24)。

data\_ram\_sel\_r[2] 为 1：选择次高 8 位字节 (23:16)。

data\_ram\_sel\_r[1] 为 1：选择次低 8 位字节 (15:8)。

data\_ram\_sel\_r[0] 为 1：选择最低 8 位字节 (7:0)。

其他情况，默认为 0。

半字数据选择 (h\_data)：

根据 data\_ram\_sel\_r 的值，选择数据 SRAM 读取的特定半字（16 位）：

data\_ram\_sel\_r[2] 为 1：选择最高 16 位半字 (31:16)。

data\_ram\_sel\_r[0] 为 1：选择最低 16 位半字 (15:0)。

其他情况，默认为 0。

字数据选择 (w\_data)：

直接使用数据 SRAM 读取的 32 位数据。

#### 5.2.7. 寄存器文件写回数据处理

寄存器文件写回数据 rf\_wdata：

根据寄存器文件结果选择信号 sel\_rf\_res 和数据 RAM 使能信号 data\_ram\_en，选择写回数据的来源：

如果 sel\_rf\_res 和 data\_ram\_en 为 1，则选择访存结果 mem\_result 作为写回数据。

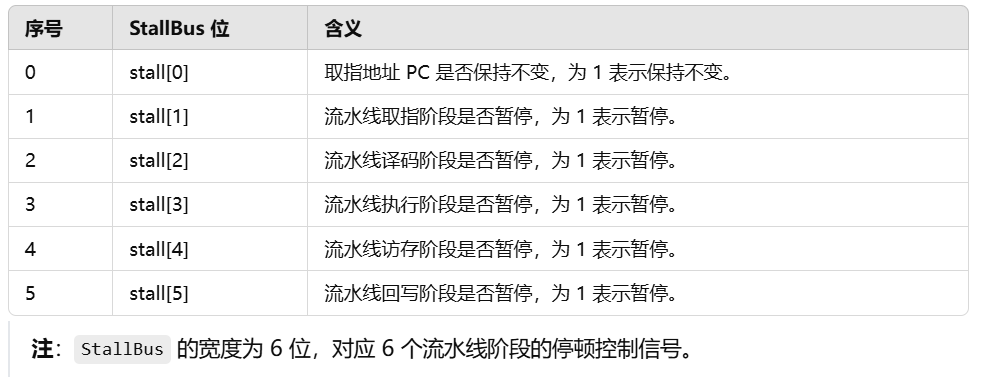
否则，使用来自 EX 阶段的运算结果 ex\_result 作为写回数据。

## 6.CTRL模块

CTRL 模块输入输出



StallBus 各位含义



CTRL 模块详细功能说明

### 6.1整体说明

CTRL 模块负责整个流水线的控制与协调，主要通过管理和生成流水线停顿（stall）信号来处理数据相关性、资源冲突和控制冒险等问题。它接收来自执行阶段（EX）、分支跳转单元（BRU）和访存阶段（MEM）的停顿请求信号，根据这些信号生成统一的停顿信号总线，以控制各个流水线阶段的暂停与恢复，确保流水线的正确性和效率。

### 6.2详细功能说明

#### 6.2.1. 输入信号与模块说明

复位信号 rst：

用于初始化控制模块。当 rst 为高电平时，CTRL 模块会清除所有停顿信号，恢复流水线到初始状态。

停顿请求信号：

stallreq\_for\_ex：来自执行阶段（EX）的停顿请求信号，通常用于处理复杂运算（如除法、多周期运算）导致的流水线停顿。

stallreq\_for\_bru：来自分支跳转单元（BRU）的停顿请求信号，通常用于处理分支指令的跳转延迟。

stallreq\_for\_load：来自访存阶段（MEM）的 Load 操作的停顿请求信号，用于处理 Load-Use 数据冒险。

停顿信号总线 stall：

用于控制流水线各阶段的暂停与恢复。CTRL 模块根据接收到的停顿请求信号，生成统一的停顿信号总线，确保各个流水线阶段协调一致地进行停顿或继续执行。

#### 6.2.2. 内部逻辑与停顿信号生成

逻辑优先级：

CTRL 模块根据不同的停顿请求信号的优先级，依次判断并设置停顿信号。当前代码中，stallreq\_for\_ex 的优先级最高，其次是 stallreq\_for\_bru，最后是 stallreq\_for\_load。

停顿信号设置：

复位 (rst 为高电平)：

当复位信号为高电平时，清除所有停顿信号，设置 stall 为 0，恢复流水线的正常运行。

处理停顿请求：

stallreq\_for\_ex 为 1：

设置 stall = 6'b001111。

具体含义：

stall[0] = 0：不保持取指地址 PC 不变。

stall[1] = 0：不暂停取指阶段。

stall[2] = 1：暂停译码阶段。

stall[3] = 1：暂停执行阶段。

stall[4] = 1：暂停访存阶段。

stall[5] = 1：暂停回写阶段。

这意味着暂停整个流水线的所有后续阶段，仅允许取指阶段继续执行。

stallreq\_for\_bru 为 1：

设置 stall = 6'b000111。

具体含义：

stall[0] = 0：不保持取指地址 PC 不变。

stall[1] = 0：不暂停取指阶段。

stall[2] = 0：不暂停译码阶段。

stall[3] = 1：暂停执行阶段。

stall[4] = 1：暂停访存阶段。

stall[5] = 1：暂停回写阶段。

这意味着暂停执行、访存和回写阶段，以处理分支跳转的相关逻辑。

当没有任何停顿请求信号为 1 时，设置 stall = 6'b000000，即不暂停任何流水线阶段，流水线正常运行。

#### 6.2.3. 停顿信号总线 stall 的具体设置

stall = 6'b001111（stallreq\_for\_ex 为 1）：

暂停译码、执行、访存和回写阶段。

允许取指阶段继续执行，确保取指阶段的数据稳定性。

stall = 6'b000111（stallreq\_for\_bru 为 1）

暂停执行、访存和回写阶段。

允许取指和译码阶段继续执行，处理分支跳转相关逻辑。

stall = 6'b000011（stallreq\_for\_load\_out 为 1）：

暂停访存和回写阶段。

允许取指、译码和执行阶段继续执行，等待 Load 操作完成。

stall = 6'b000000（无任何停顿请求）：

不暂停任何流水线阶段，流水线正常运行。

## 7.HI\_LO\_REG模块

HI\_LO\_REG 模块输入输出

HI\_LO\_REG 模块详细功能说明

### 7.1整体说明

hi\_lo\_reg 模块负责管理特殊寄存器 HI 和 LO，这两个寄存器通常用于存储乘法和除法运算的结果。具体来说：

HI 寄存器：存储乘法运算的高 32 位结果或除法运算的余数。

LO 寄存器：存储乘法运算的低 32 位结果或除法运算的商。

此外，hi\_lo\_reg 模块支持独立读取和写入 HI 和 LO 寄存器，通过指令 MFHI（Move From HI）、MFLO（Move From LO）、MTHI（Move To HI）和 MTLO（Move To LO）进行操作。

### 7.2详细功能说明

#### 7.2.1. 输入信号与模块说明

时钟信号 clk：

同步时钟信号，驱动模块内部的时序逻辑。

停顿信号 stall：

用于控制流水线的停顿。虽然该信号作为输入存在，但在当前模块中并未被使用。根据设计需求，未来可以根据停顿信号的状态调整 HI/LO 寄存器的行为。

HI/LO 寄存器写使能信号 hi\_we 和 lo\_we：

hi\_we：当为 1 时，允许将 hi\_wdata 写入 HI 寄存器。

lo\_we：当为 1 时，允许将 lo\_wdata 写入 LO 寄存器。

HI/LO 寄存器写入数据 hi\_wdata 和 lo\_wdata：

hi\_wdata：要写入 HI 寄存器的数据。

lo\_wdata：要写入 LO 寄存器的数据。

HI/LO 寄存器读出数据 hi\_rdata 和 lo\_rdata：

hi\_rdata：从 HI 寄存器读取的数据。

lo\_rdata：从 LO 寄存器读取的数据。

#### 7.2.2. 内部寄存器与数据存储

内部寄存器 reg\_hi 和 reg\_lo：

reg\_hi：32 位寄存器，用于存储 HI 寄存器的当前值。

reg\_lo：32 位寄存器，用于存储 LO 寄存器的当前值。

3. 时序逻辑

寄存器更新逻辑：

在每个时钟上升沿，根据写使能信号更新 HI 和 LO 寄存器的值：

同时使能 hi\_we 和 lo\_we：

reg\_hi <= hi\_wdata：更新 HI 寄存器。

reg\_lo <= lo\_wdata：更新 LO 寄存器。

仅使能 lo\_we：

reg\_lo <= lo\_wdata：仅更新 LO 寄存器。

仅使能 hi\_we：

reg\_hi <= hi\_wdata：仅更新 HI 寄存器。

既不使能 hi\_we 也不使能 lo\_we：

保持 HI 和 LO 寄存器的当前值不变。

#### 7.2.3. HI/LO 寄存器的读出

读出逻辑：

hi\_rdata 和 lo\_rdata 直接从内部寄存器 reg\_hi 和 reg\_lo 赋值，确保外部模块能够读取到当前的 HI 和 LO 寄存器值。

# 总结与思考

刘庆然：首先先说说实验的建议。主要问题在于没有一个较好的演示，虽然学长所发的各种资料已经非常全面，涵盖了所有需要用到的相关知识，但是由于个人基础较差，通过所发的资料来捋明白整个流程该怎么做就已经耗费了很长时间。所以建议在以后的实验中，可以给出比如从基本代码导入后所报错误，如何查询，如何修改，然后不用给的太多，到给出通过第一个点的流程即可，这样让同学们做起来也更好上手，在第一个点跟着做的过程中就能大致知道该怎么做了

另外在本次实验中，我深入学习并实践了基于 Verilog 的 CPU 流水线设计，涵盖了取指（IF）、译码（ID）、执行（EX）、访存（MEM）和写回（WB）等关键模块的开发与优化。通过编写和分析 MEM、WB、CTRL、hi\_lo\_reg 及 IF 模块的代码，我掌握了流水线各阶段的功能实现、数据传递及控制信号管理。同时，我解决了如 Load-Use 数据冒险、分支跳转等常见的流水线冲突问题，提升了整体设计的稳定性和效率。尽管整体上还是不能很好的把握，但对CPU已经有了初步的认识，手搓CPU这样一个听起来很高大上的词也变得具体起来。

马智鸿：实验中，我主要负责部分模块的实现和报告的编写工作，通过实验，我进一步理解了 CPU 五级流水线（IF、ID、EX、MEM、WB）的具体实现原理及各阶段的功能。例如，IF 阶段如何准确地获取下一条指令地址，ID 阶段的指令译码逻辑，以及 EX 阶段复杂的算术逻辑运算过程。这些知识以前可能停留在理论层面，而通过动手实践后，我能够直观地感受到这些模块之间的紧密联系和协同工作的重要性。实验任务较为复杂，幸运的是有刘庆然队长，明确的分工和高效的沟通使我们的项目稳步推进。在实验过程中，我们遇到了许多挑战，例如访存阶段的 Load-Use 数据冒险处理，以及地址转换中虚拟地址和物理地址的对应关系调试。但队友见共同查阅资料、调试修改，棘手的问题也能解决掉。感谢这次cpu模拟制作实验，让我更有信心面对接下来的挑战！