实验报告成绩:	成绩评定日期:

2022 ~ 2023 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 2201（未来实验班）

组长：王新宇

组员：刘力瑞

报告日期：2024.1.5

目录

1. 实验任务与分工

1.1 实验任务概述

本实验中要完成 CPU 流水线的测试，需要深入理解计算机系统的硬件运行机制，掌握流水线技术在提高 CPU 性能方面的原理和应用。要构建构建一个包含五级流水线，需要使用 verilog 完成取指 IF、译码 ID、执行 EX、访存 MEM、写回 WB 的 CPU 模型等代码，以实现了一个具备基本功能的 CPU 流水线系统。

1.2 小组成员分工详情

姓名	任务分工	任务量占比
王新宇	实现 IF，ID，MEM，WB,HILO，CTRL,STALL 等指令以及主要代码编写，参加实验报告编写	70%
刘力瑞	主要负责在流水线中的 EX 阶段编写，乘法器的设计参与实验报告的编写	30%

2. 实验设计与实现

2.1 CPU 总体架构设计

CPU 总体围绕一条五级流水线展开，包含多个功能模块，各模块协同工作实现指令的取指、译码、执行、访存和写回等操作，同时具备对流水线的控制以及特殊寄存器的处理功能，具体架构可以分为以下部分：

1. 流水线模块：包括 IF（取指）、ID（译码）、EX（执行）、MEM（访存）、WB（写回）模块，构建了 CPU 指令执行的五级流水线基本框架，负责指令在不同阶段的处理和流转。

2.运算与寄存器模块：ALU 相关位于/lib 目录下的 alu.v 等文件构建了 ALU，用于执行算术和逻辑运算，为 EX 模块提供运算支持。

3.寄存器相关：regfile.v 构建了通用寄存器，hi_lo_reg.v 实现了 hilo 寄存器（用于乘法和除法结果的存储等），这些寄存器在指令执行过程中用于数据的存储和传递。

4.控制模块：CTRL 模块接收各阶段的请求信号，控制流水线各阶段的暂停和运行，协调各模块工作节奏。

5.数据传输总线：各模块之间通过特定宽度的数据总线连接，如 IF 模块到 ID 模块通过 if_to_id_bus（33 位）传输数据，ID 模块到 EX 模块通过 id_to_ex_bus（159 位）传输数据等，确保数据在不同阶段的准确传递。

6.指令处理流程：

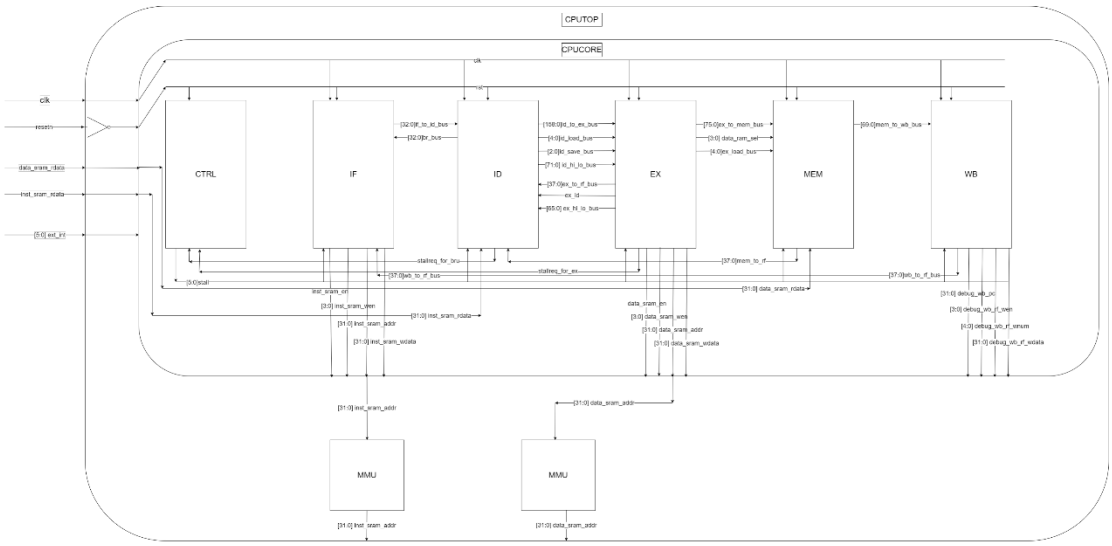
取指（IF）：从指令内存中获取指令，根据时钟、复位、暂停和跳转等信号控制指令地址的生成和指令的读取，将读取的指令传递给 ID 模块。

译码（ID）：对指令进行译码，判断指令类型（一般指令或跳转指令等），读取通用寄存器的值，处理数据相关，确定操作数来源，将译码结果和相关数据传递给 EX 模块，同时处理流水线暂停请求。

执行（EX）：执行运算操作，包括算术、逻辑运算，计算地址，处理访存请求，将运算结果和访存相关信息传递给 MEM 模块，并向 ID 模块反馈部分数据。

访存（MEM）：根据 EX 模块传来的地址进行内存访问操作，处理 load 和 store 指令，将访存结果传递给 WB 模块。

写回（WB）：将结果写回寄存器堆，完成指令执行的最后一步。



2.2 代码文件概述

项目包含多个模块文件，共同构建了一条流水线的基本框架以及相关运算和寄存器等功能。包括 IF.v、ID.v、EX.v、MEM.v、WB.v、hi_lo_reg.v、mycpu_core.v、mycpu_top.v 等文件搭建流水线框架；位于 /lib 目录下的 alu.v、decoder_2_4.v、decoder_5_32.v、decoder_6_64.v、defines.vh、div.v、mmu.v、regfile.v 构建 ALU 和寄存器并定义头文件；位于 /lib/mul 目录下的 add.v、fa.v、mul.v 实现乘法运算。

2.3 测试环境与方法

测试平台：装有 Vivado 的 windows 系统。

使用 VSCode 编写代码，提供代码编辑功能。

利用 Vivado 进行模拟仿真，用于验证设计的正确性。

借助 git 进行版本管理，便于代码的更新、回溯和团队协作。

通过 GitHub 搭建项目仓库，实现代码的存储、共享和团队间的协同开发。

测试方法

debug

在实验的 debug 过程中，通过在波形图中添加可能有问题的数值，查看提示 pc 值附近目标的波形图来定位出错的位置及原因。通过观察波形图中信号的变化、数据的传输以及各模块之间的交互情况，分析指令执行过程中是否存在异常，如数据错误、信号时序问题等，从而找出导致错误的具体指令或模块操作，以便进行针对性的修改和优化。

2.4 完成指令

运算指令	ADD	ADDI	ADDU	ADDIU
	SUB	SUBU	SLT	SLTI
	SLTU	SLTIU	DIV	DIVU
	MULT	MULTU		
逻辑运算指令	AND	ANDI	LUI	NOR
	OR	ORI	XOR	XORI
移位指令	SLL	SLLV	SRA	SRAV
	SRL	SRLV		
跳转指令	BEQ	BNE	BGEZ	BGTZ
	BLEZ	BLTZ	BLTZAL	BGEZAL
	J	JAL	JR	JALR
数据移动指令	MFHI	MFLO	MTHI	MTLO
访存指令	LB	LBU	LH	LHU
	LW	SB	SH	SW

3 流水线各阶段详细设计

3.1 IF（取指令）阶段

IF（指令获取）段是 CPU 流水线的第一个阶段，主要负责从内存中获取指令。

具体功能包括：

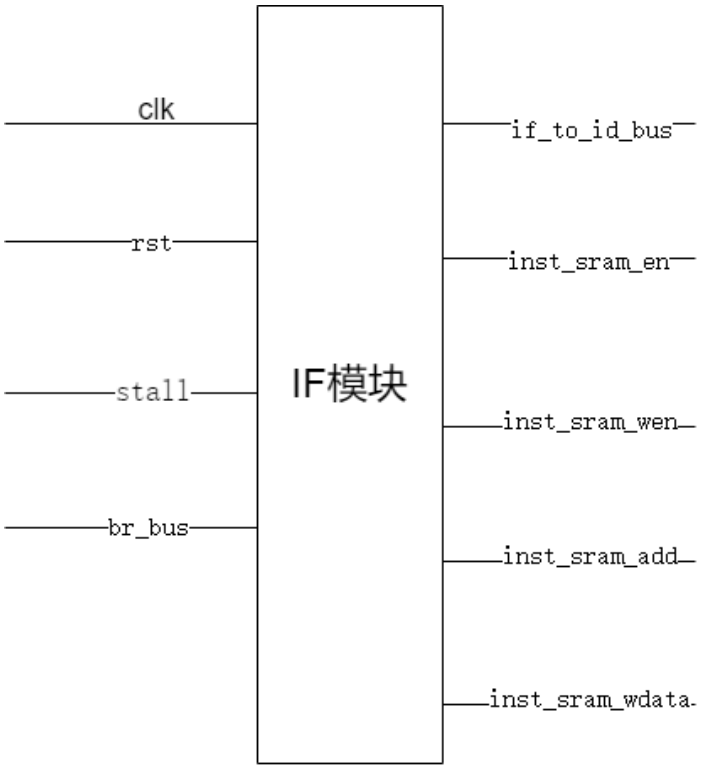
1. **获取下一条指令**：根据当前的程序计数器（PC，Program Counter）值从指令存储器（SRAM）中获取指令。
2. **更新 PC 值**：通过计算或跳转控制更新程序计数器的值，确保 CPU 始终获取正确的下一条指令。
3. **跳转支持**：如果当前执行的是跳转指令（如 beq，jal 等），根据分支控制信号更新 PC 值，进行跳转。
4. **信号传递**：将当前 PC 值和相关控制信号传递给下一级的 ID（指令译码）阶段。

端口设置：

端口名称	方向	类型	位宽	描述
clk	输入	wire	1	时钟信号
rst	输入	wire	1	复位信号
stall	输入	wire	StallBus	暂停信号
br_bus	输入	wire	BR_WD	分支跳转信息
if_to_id_bus	输出	wire	IF_TO_ID_WD	传给 ID 阶段的指令数据
inst_sram_en	输出	wire	1	指令存储器使能信号
inst_sram_wen	输出	wire	4	指令存储器写使能信号
inst_sram_addr	输出	wire	32	指令存储器地址
inst_sram_wdata	输出	wire	32	指令存储器写数据

信号名称	类型	位宽	描述
pc_reg	reg	32	当前程序计数器值
ce_reg	reg	1	使能信号
next_pc	wire	32	下一周期的 PC 值
br_e	wire	1	是否有跳转
br_addr	wire	32	跳转地址

最终的图像结构显示为



3.2 ID（指令译码）阶段

该模块是 MIPS 处理器的 ID（译码）阶段，负责从 IF（取指）阶段接收指令，并进行指令译码、寄存器读取、数据相关性处理，以及生成控制信号传送到执行阶段（EX）。此外，该模块还处理来自后续阶段（EX、MEM、WB）的写回数据，进行数据转发，以减少数据冲突。

1. **寄存器文件 (u_regfile)**:

读取两个寄存器数据 (rdata1, rdata2)。

根据写回信号 (wb_rf_we, wb_rf_waddr, wb_rf_wdata) 写入数据。

2. **HI/LO 寄存器 (u_hi_lo_reg)**:

读取和写入 HI/LO 寄存器数据。

处理乘法和除法指令的结果。

3. **译码器 (u0_decoder_6_64, u1_decoder_6_64, etc.)**:

将指令的操作码和功能码译码为控制信号。

4. **数据转发逻辑**:

处理来自 EX、MEM、WB 阶段的写回数据，进行数据转发，避免数据相关性冲突。

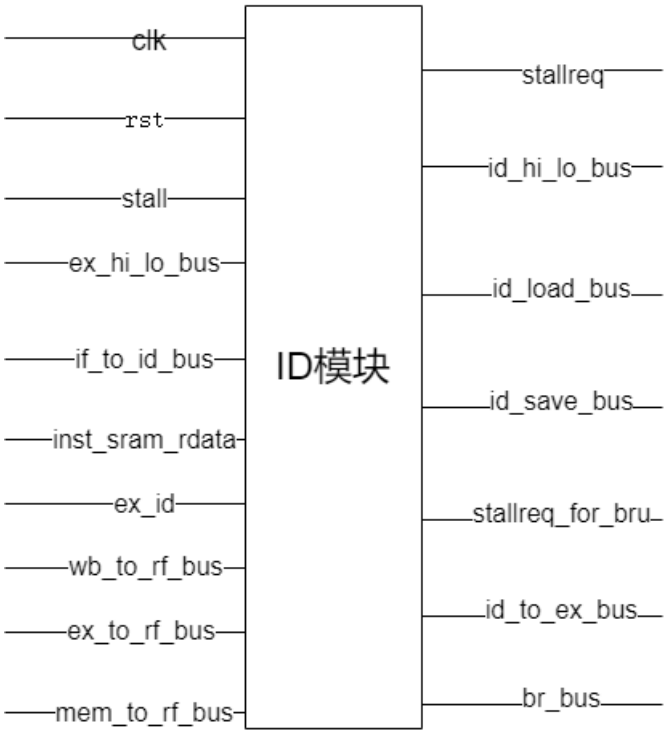
5. **分支逻辑**:

生成分支指令的目标地址（br_addr）和分支使能信号（br_e)。

端口名称	方向 位宽	描述
clk	输入 1	时钟信号
rst	输入 1	复位信号
stall	输入 StallBus	暂停信号
if_to_id_bus	输入 IF_TO_ID_WD	从 IF 阶段接收的数据总线
inst_sram_rdata	输入 32	指令存储器读取的数据
ex_id	输入 1	来自执行阶段的分支指令信号
wb_to_rf_bus	输入 WB_TO_RF_WD	来自写回阶段的写回总线
ex_to_rf_bus	输入 EX_TO_RF_WD	来自执行阶段的写回总线
mem_to_rf_bus	输入 MEM_TO_RF_WD	来自内存阶段的写回总线
ex_hi_lo_bus	输入 66	来自执行阶段的 HI/LO 寄存器数据
id_load_bus	输出 LoadBus	负载指令信号总线
id_save_bus	输出 SaveBus	存储指令信号总线
stallreq_for_bru	输出 1	向分支预测单元请求暂停信号
id_to_ex_bus	输出 ID_TO_EX_WD	传送到执行阶段的总线
br_bus	输出 BR_WD	分支指令相关信号总线

信号名称	位宽	描述
inst	32	当前指令
id_pc	32	当前指令的 PC 值
rs, rt, rd	5	指令中的寄存器地址
imm, offset	16	指令中的立即数或偏移量
alu_op	12	ALU 操作控制信号
rf_we	1	寄存器文件写使能信号
rf_waddr	5	寄存器文件写地址
ndata1, ndata2	32	读取的寄存器数据（经过数据转发）
br_e	1	分支指令使能信号
br_addr	32	分支目标地址

最终图像结构表示为：



3.3 EX（执行）阶段

输入端口:

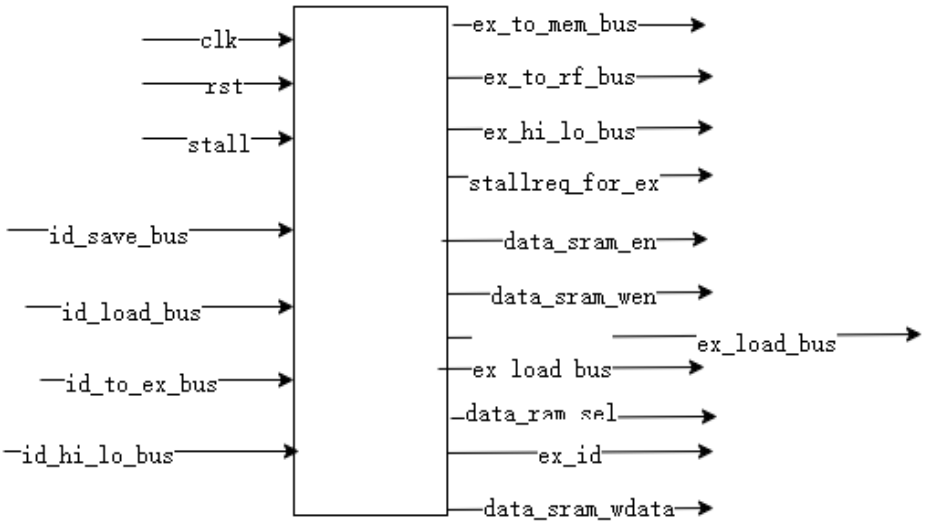
接口名	宽度	数据
clk	1	时钟信号
rst	1	复位信号
stall	6	控制暂停信号
id_to_ex_bus	169	ID 段至 EX 段数据
id_load_bus	5	ID 段传递读的数据
id_save_bus	3	ID 段传递写的数据
id_hi_lo_bus	72	ID 段至 hilo 段的数据

输出端口:

接口名	宽度	数据
ex_to_mem_bus	80	EX 段传给 MEM 段数据
ex_to_rf_bus	38	EX 至 regfile 的数据
ex_hi_lo_bus	66	EX 段至 hilo 段的数据
stallreq_for_ex	1	除法暂停请求信号
data_sram_en	1	访问内存与否
data_sram_wen	4	选择写入字节操作
data_sram_addr	32	内存数据存放的地址
data_sram_wdata	32	要写入内存的数据

ex_id	38	EX 段传给 ID 段的数据
data_ram_sel	4	内存字节选择信号
ex_load_bus	5	EX 段读取的数据

相关输入输出接口图示



ALU 运算:

id_to_ex_bus 组成

信号名称	位宽	描述
ex_pc	32	位 程序计数器（PC）值
inst	32	位 当前指令
alu_op	12	位 ALU 操作码，指定 ALU 执行的操作
sel_alu_src1	3	位 操作数 1 的来源选择信号
sel_alu_src2	4	位 操作数 2 的来源选择信号
data_ram_en	1	位 是否访问内存的标志
data_ram_wen	4	位 内存写操作的控制信号
rf_we	1	位 是否写回寄存器文件的标志
rf_waddr	5	位 写回寄存器的地址
sel_rf_res	1	位 选择写回寄存器的数据源
rf_rdata1	32	位 寄存器文件中第一个操作数的数据
rf_rdata2	32	位 寄存器文件中第二个操作数的数据

```
alu u_alu(
    .alu_control (alu_op          ),//alu_op 是控制信号，表示进行的操作
```

```
        .alu_src1       (alu_src1       ),
        .alu_src2       (alu_src2       ),
        .alu_result     (alu_result     )
    );
    wire mul_signed; // 有符号乘法标记
```

1. 则是有乘法与否。乘法运算是通过模拟二进制乘法的逐位加法和左移操作来实现的，使用了 逐位加法 和 左移操作 来实现乘法的核心部分。这个过程是基于二进制乘法的原理。逐位加法： 如果操作数的当前位为 1，则将 op1 加到结果 结果位 上。这样就模拟了二进制乘法中乘数对被乘数的加法操作。左移： 在每次处理完 当前位 之后，op1 会被左移一位，以便模拟每一位的乘法。
2. 乘法结束处理，在乘法过程结束时，如果是有符号乘法并且结果需要调整（即 操作数 1 和操作数 2 符号不同），代码会对结果进行补码处理（取反加一），以得到正确的符号。

具体步骤

1. **初始化：**

如果 进入除法状态且 取消乘法位 0，且操作数 1 和 操作数 2 都不为零，进入状态乘法计算状态。

如果任一操作数为零，操作数为零处理状态。

2. **符号扩展与补码转换：**

如果是有符号乘法，且操作数为负数（符号位为 1），则：对操作数 1 和操作数 2 进行补码转换。补码转换是通过对操作数取反并加 1 来实现。

3. **逐位乘法：**进入状态 乘法计算状态，开始逐位乘法计算。通过计数器 cnt 控制对 op2 每一位的处理：如果 操作数 2 当前数 为 1，则将 操作数 1 加到 结果上。每次加完后，将 op1 左移一位，准备下一位的乘法。计数器 cnt 自增，直到所有位都处理完。
4. **结束处理：**在计数器 cnt 达到预定值（即 32 位乘法处理完成），如果是有符号乘法，且操作数符号不同（即结果应该为负），对结果 result_o 进行补码转换（取反加一）。进入状态乘法完成。
5. **返回初始化状态：**

在状态乘法完成后，如果如果乘法开始状态位 0，返回初始化状态，准备开始下一次操作。

总结

乘法实现的关键在于逐位加法和左移操作：

对于每一位的乘法，检查 操作数 2 当前位 是否为 1，如果是，则将 op1 加到结果中。

每次处理完一位，op1 左移一位，相当于将操作数 1 乘以对应的位权。

最后根据符号位，调整结果的符号，并标记运算结束。

这种实现方式模拟了二进制的逐位乘法计算，适用于支持有符号和无符号整数的乘法运算。

3.4 MEM（访存）阶段

该模块为MIPS处理器的MEM阶段，负责处理内存访问操作。它从EX阶段接收数据，并根据指令类型进行内存读取或写入操作。该模块还负责将内存读取的数据格式化，并将其传递到WB阶段。

1. **寄存器模块**:

保存来自EX阶段的数据总线、字节选择信号和加载指令信号。

2. **指令解码模块**:

将 ex_load_bus_r 解码为具体的加载指令信号（inst_lb, inst_lbu, inst_lh, inst_lhu, inst_lw）。

3. **数据路径模块**:

根据 data_ram_sel_r 选择从内存读取的字节、半字或字数据，并根据加载指令类型进行格式化（如符号扩展或零扩展）。

4. **结果选择模块**:

根据 sel_rf_res 和 data_ram_en 选择 rf_wdata 是来自 mem_result 还是 ex_result。

5. **数据打包模块**:

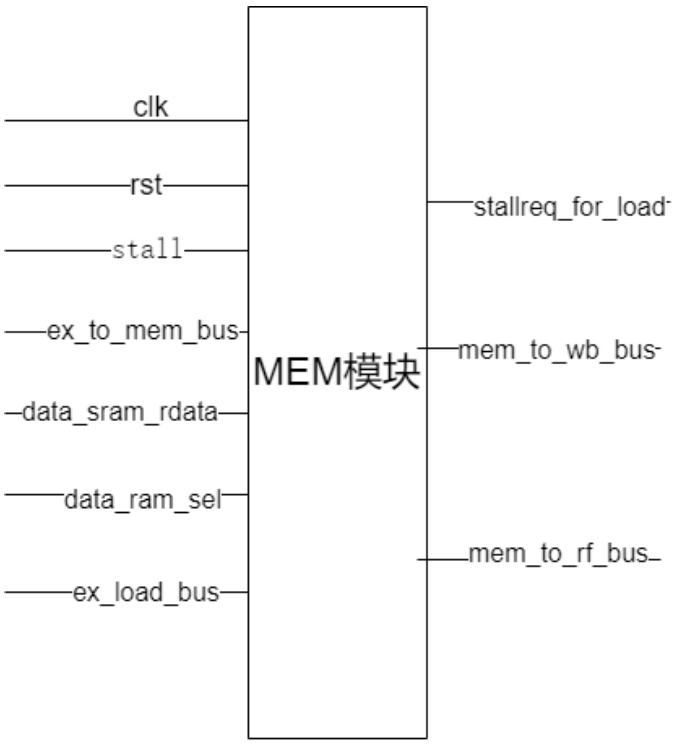
将 mem_pc、rf_we、rf_waddr 和 rf_wdata 打包成 mem_to_wb_bus 和 mem_to_rf_bus，传递到后续阶段。

端口介绍：

端口名称	方向	位宽	描述
clk	输入	1	时钟信号
rst	输入	1	复位信号
stall	输入	StallBus	暂停信号
ex_to_mem_bus	输入	EX_TO_MEM_WD	来自EX阶段的数据总线
data_sram_rdata	输入	32	内存读取的数据
data_ram_sel	输入	4	字节选择信号，用于选择从内存读取的数据的字节
ex_load_bus	输入	LoadBus	来自EX阶段的加载指令信号
stallreq_for_load	输出	1	因加载指令产生的暂停请求信号
mem_to_wb_bus	输出	MEM_TO_WB_WD	传送到WB阶段的数据总线
mem_to_rf_bus	输出	MEM_TO_RF_WD	传送到寄存器文件的数据总线

信号介绍：

信号名称	位宽	描述
mem_pc	32	当前 PC 地址
data_ram_en	1	数据使能信号，控制内存读取
data_ram_wen	4	数据写使能信号，选择要写入的字节
sel_rf_res	1	选择寄存器文件写数据的来源
rf_we	1	寄存器文件写使能信号
rf_waddr	5	寄存器文件写地址
rf_wdata	32	寄存器文件写数据
ex_result	32	来自 EX 阶段的结果数据
mem_result	32	根据加载指令类型格式化的内存数据
inst_lb, inst_lbu, inst_lh, inst_lhu, inst_lw	1 each	各种加载指令的使能信号
b_data, h_data, w_data	8, 16, 32	从内存读取的字节、半字和字数据



3.5 WB（写回）阶段

该模块是 MIPS 流水线的写回（Write Back，WB）阶段。其主要功能是从内存到写回（MEM_TO_WB）阶段的流水线寄存器中读取数据，并将这些数据写回到寄存器堆（Register File，RF）。此外，该模块还提供了调试信号，以便在调试时观察内部状态。

主要有以下功能模块：

1. **寄存器存储**:

使用寄存器 mem_to_wb_bus_r 存储从 mem_to_wb_bus 接收到的数据。

根据 rst 和 stall 信号控制寄存器的复位和更新。

2. **信号提取**:

从 mem_to_wb_bus_r 中提取 wb_pc、rf_we、rf_waddr 和 rf_wdata。

3. **数据传递**:

将提取的信号组合成 wb_to_rf_bus，传给寄存器堆。

4. **调试信号**:

提供调试信号 debug_wb_pc、debug_wb_rf_wen、debug_wb_rf_wnum 和 debug_wb_rf_wdata，方便观察内部状态。

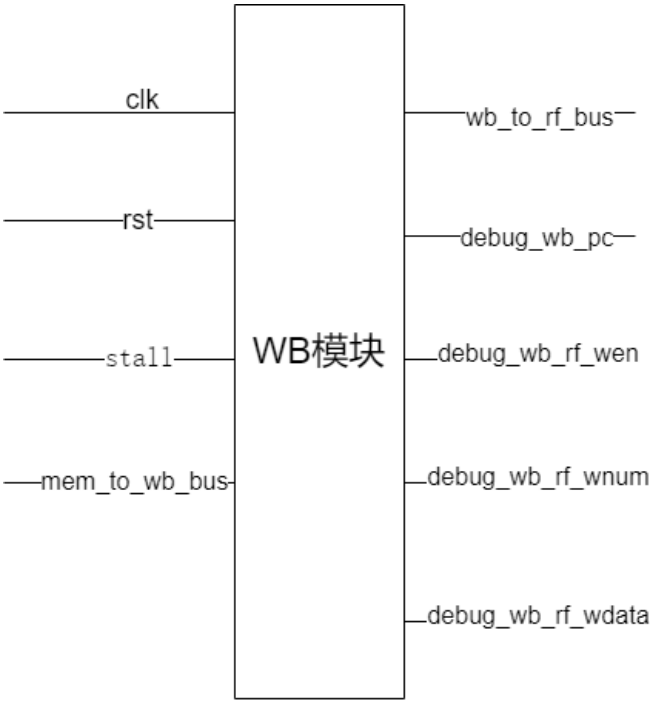
端口介绍：

端口名称	**方向**	**宽度**	**描述**
clk	输入	1	时钟信号
rst	输入	1	复位信号，高有效
stall	输入	StallBus-1:0	流水线停顿信号，多位信号
mem_to_wb_bus	输入	MEM_TO_WB_WD-1:0	来自内存阶段的数据总线
wb_to_rf_bus	输出	WB_TO_RF_WD-1:0	传给寄存器堆的总线
debug_wb_pc	输出	32	调试用程序计数器
debug_wb_rf_wen	输出	4	调试用寄存器写使能信号
debug_wb_rf_wnum	输出	5	调试用寄存器写地址
debug_wb_rf_wdata	输出	32	调试用寄存器写数据

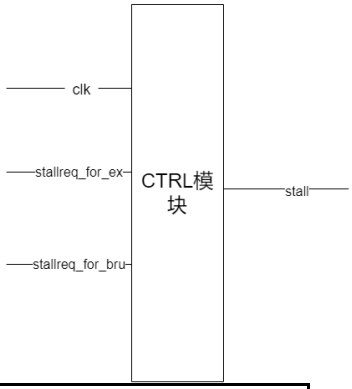
信号介绍：

信号名称	**宽度**	**描述**
mem_to_wb_bus_r	MEM_TO_WB_WD-1:0	存储从 mem_to_wb_bus 接收到的数据的寄存器
wb_pc	32	程序计数器，32 位

信号名称	宽度	描述
rf_we	1	寄存器写使能，1 位
rf_waddr	5	寄存器写地址，5 位
rf_wdata	32	寄存器写数据，32 位



2.6 CTRL 模块



整体说明：

接收各段传递过来的流水线请求信号，从而控制流水线各阶段的运行。接口如右图所示。

序号	接口名	宽	输入/输出	作用
1	clk	1	输入	时钟信号
2	stallreq_for_ex	1	输入	执行阶段的指令是否请求流水线暂停
3	stallreq_for_bru	5	输入	Load 命令是否请求流水线暂停
4	stall	6	输出	暂停信号

功能说明：

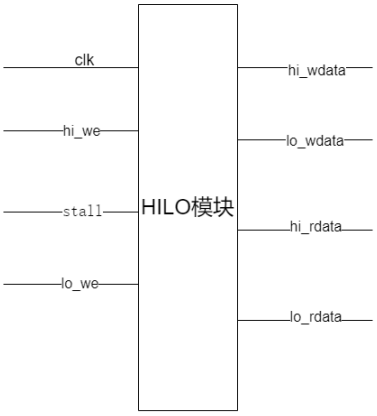
假设位于流水线第 n 阶段的指令需要多个周期，进而请求流水线暂停，那么需要保持取指令地址 PC 不变，同时保持流水线第 n 阶段及之前的各个阶段

的寄存器保持不变，而第 n 阶段后面的指令继续运行。stall[0]为 1 表示没有暂停，1-5 为 1 时分别 代表 if 段、id 段、ex 段、 mem 段、wb 段暂停。

2.7 HILO 寄存器模块

整体说明：

hi 和 lo 属于协处理器，不在通用寄存器的范围内，这两个寄存器主要是在用来处理乘法和除法。以乘法作为示例，如果两个整数相乘，那么乘法的结果低位保存在 lo 寄存器，高位保存在 hi 寄存器。 当然，这两个寄存器也可以独立进行读取和写入。读的时候，使用 mfhi、mflo；写入的时候，用 mthi、mtlo。和通用寄存器不同，mfhi、mflo 是在执行阶段才开始从 hi、lo 寄存器获取数值的。写入则和通用寄存器一样，也是在写回的时候完成的。



可以直接改写 lib 下的 regfile.v，也可以添加 hiloreg.v，创建 u_hi_lo_reg，但是 MEM、WB 也要跟着改，这里我们采用第二种方法，即添加 hiloreg.v 文件。接口如右图所示。

表 1 HILO 寄存器输入输出

序号	接口名	宽度	输入/输出	作用
1	clk	1	输入	时钟信号
2	stall	6	输入	控制暂停信号
3	hi_we	1	输入	hi 寄存器的写使能信号
4	lo_we	1	输入	lo 寄存器的写使能信号
5	hi_wdata	32	输出	Hi 寄存器写的数据
6	lo_wdata	32	输出	Lo 寄存器写的数据
7	hi_rdata	32	输出	Hi 寄存器读的数据
8	lo_rdata	32	输出	Lo 寄存器读的数据

功能说明：

当 hi_we 和 lo_we 均为 1 时，寄存器 reg_hi 和 reg_lo 同时将 hi_wdata 和 lo_wdata 写入。当 hi_we 为 0，lo_we 为 1 时，reg_lo 将 lo_wdata 写入；当 hi_we 为 1，lo_we 为 0 时，reg_hi 将 hi_wdata 写入。hi_rdata 和 lo_rdata 分别输出 reg_hi 和 reg_lo 中的数据。

4. 实验测试与结果分析

4.1 测试结果展示

```
Test begin!
----[    14025 ns] Number 8'd01 Functional Test Point PASS!!!
          [    22000 ns] Test is running, debug_wb_pc = 0xbfc5e4d4
          [    32000 ns] Test is running, debug_wb_pc = 0xbfc5f474
----[    40475 ns] Number 8'd02 Functional Test Point PASS!!!
          [    42000 ns] Test is running, debug_wb_pc = 0xbfc89440
----[    49355 ns] Number 8'd03 Functional Test Point PASS!!!
          [    52000 ns] Test is running, debug_wb_pc = 0xbfc3ad58
          [    62000 ns] Test is running, debug_wb_pc = 0xbfc3c260
----[    71115 ns] Number 8'd04 Functional Test Point PASS!!!
          [    72000 ns] Test is running, debug_wb_pc = 0xbfc23898
          [    82000 ns] Test is running, debug_wb_pc = 0xbfc24c40
          [    92000 ns] Test is running, debug_wb_pc = 0xbfc2621c
          [ 102000 ns] Test is running, debug_wb_pc = 0xbfc2776c
----[ 104845 ns] Number 8'd05 Functional Test Point PASS!!!
          [ 112000 ns] Test is running, debug_wb_pc = 0xbfc4a0ac
----[ 117885 ns] Number 8'd06 Functional Test Point PASS!!!
          [ 122000 ns] Test is running, debug_wb_pc = 0xbfc6a68c
          [ 132000 ns] Test is running, debug_wb_pc = 0xbfc6b62c
          [ 142000 ns] Test is running, debug_wb_pc = 0xbfc6c5cc
----[ 144265 ns] Number 8'd07 Functional Test Point PASS!!!
          [ 152000 ns] Test is running, debug_wb_pc = 0xbfc509e4
          [ 162000 ns] Test is running, debug_wb_pc = 0xbfc51984
----[ 167675 ns] Number 8'd08 Functional Test Point PASS!!!
          [ 172000 ns] Test is running, debug_wb_pc = 0xbfc03bb0
          [ 182000 ns] Test is running, debug_wb_pc = 0xbfc04b50
----[ 185575 ns] Number 8'd09 Functional Test Point PASS!!!
          [ 192000 ns] Test is running, debug_wb_pc = 0xbfc3e008
          [ 202000 ns] Test is running, debug_wb_pc = 0xbfc3efa8
----[ 203475 ns] Number 8'd10 Functional Test Point PASS!!!
          [ 212000 ns] Test is running, debug_wb_pc = 0xbfc6f800
          [ 222000 ns] Test is running, debug_wb_pc = 0xbfc707a0
----[ 222735 ns] Number 8'd11 Functional Test Point PASS!!!
          [ 232000 ns] Test is running, debug_wb_pc = 0xbfc02648
----[ 237365 ns] Number 8'd12 Functional Test Point PASS!!!
          [ 242000 ns] Test is running, debug_wb_pc = 0xbfc3faec
          [ 252000 ns] Test is running, debug_wb_pc = 0xbfc40f3c
----[ 261915 ns] Number 8'd13 Functional Test Point PASS!!!
```

[262000 ns] Test is running, debug_wb_pc = 0xbfc00d58
[272000 ns] Test is running, debug_wb_pc = 0xbfc64ab8
[282000 ns] Test is running, debug_wb_pc = 0xbfc65db8
[292000 ns] Test is running, debug_wb_pc = 0xbfc67094
----[296425 ns] Number 8'd14 Functional Test Point PASS!!!
[302000 ns] Test is running, debug_wb_pc = 0xbfc84484
[312000 ns] Test is running, debug_wb_pc = 0xbfc85814
[322000 ns] Test is running, debug_wb_pc = 0xbfc86b8c
----[330855 ns] Number 8'd15 Functional Test Point PASS!!!
[332000 ns] Test is running, debug_wb_pc = 0xbfc7a0f0
----[333825 ns] Number 8'd16 Functional Test Point PASS!!!
----[336795 ns] Number 8'd17 Functional Test Point PASS!!!
----[338455 ns] Number 8'd18 Functional Test Point PASS!!!
----[340625 ns] Number 8'd19 Functional Test Point PASS!!!
[342000 ns] Test is running, debug_wb_pc = 0xbfc87f18
----[342805 ns] Number 8'd20 Functional Test Point PASS!!!
[352000 ns] Test is running, debug_wb_pc = 0xbfc8006c
[362000 ns] Test is running, debug_wb_pc = 0xbfc8100c
----[364615 ns] Number 8'd21 Functional Test Point PASS!!!
[372000 ns] Test is running, debug_wb_pc = 0xbfc0b1c8
[382000 ns] Test is running, debug_wb_pc = 0xbfc0c168
----[385015 ns] Number 8'd22 Functional Test Point PASS!!!
[392000 ns] Test is running, debug_wb_pc = 0xbfc332a8
[402000 ns] Test is running, debug_wb_pc = 0xbfc34248
----[406375 ns] Number 8'd23 Functional Test Point PASS!!!
[412000 ns] Test is running, debug_wb_pc = 0xbfc61798
[422000 ns] Test is running, debug_wb_pc = 0xbfc62738
[432000 ns] Test is running, debug_wb_pc = 0xbfc636d8
----[432785 ns] Number 8'd24 Functional Test Point PASS!!!
[442000 ns] Test is running, debug_wb_pc = 0xbfc7b564
[452000 ns] Test is running, debug_wb_pc = 0xbfc7c504
----[456195 ns] Number 8'd25 Functional Test Point PASS!!!
[462000 ns] Test is running, debug_wb_pc = 0xbfc4d470
[472000 ns] Test is running, debug_wb_pc = 0xbfc4e410
----[481535 ns] Number 8'd26 Functional Test Point PASS!!!
[482000 ns] Test is running, debug_wb_pc = 0xbfc6cfc8
[492000 ns] Test is running, debug_wb_pc = 0xbfc6df68
----[499435 ns] Number 8'd27 Functional Test Point PASS!!!
[502000 ns] Test is running, debug_wb_pc = 0xbfc8a360
[512000 ns] Test is running, debug_wb_pc = 0xbfc8b300
[522000 ns] Test is running, debug_wb_pc = 0xbfc8c2a0
----[525825 ns] Number 8'd28 Functional Test Point PASS!!!
[532000 ns] Test is running, debug_wb_pc = 0xbfc78914
[542000 ns] Test is running, debug_wb_pc = 0xbfc798b4

```
----[ 546225 ns] Number 8'd29 Functional Test Point PASS!!!
          [ 552000 ns] Test is running, debug_wb_pc = 0xbfc475a4
          [ 562000 ns] Test is running, debug_wb_pc = 0xbfc48544
          [ 572000 ns] Test is running, debug_wb_pc = 0xbfc494e4
----[ 572635 ns] Number 8'd30 Functional Test Point PASS!!!
          [ 582000 ns] Test is running, debug_wb_pc = 0xbfc09260
          [ 592000 ns] Test is running, debug_wb_pc = 0xbfc0a200
----[ 593035 ns] Number 8'd31 Functional Test Point PASS!!!
          [ 602000 ns] Test is running, debug_wb_pc = 0xbfc76b40
          [ 612000 ns] Test is running, debug_wb_pc = 0xbfc77ae0
----[ 615165 ns] Number 8'd32 Functional Test Point PASS!!!
          [ 622000 ns] Test is running, debug_wb_pc = 0xbfc42d9c
          [ 632000 ns] Test is running, debug_wb_pc = 0xbfc43d3c
----[ 634365 ns] Number 8'd33 Functional Test Point PASS!!!
          [ 642000 ns] Test is running, debug_wb_pc = 0xbfc0d4ac
          [ 652000 ns] Test is running, debug_wb_pc = 0xbfc0e44c
----[ 656735 ns] Number 8'd34 Functional Test Point PASS!!!
          [ 662000 ns] Test is running, debug_wb_pc = 0xbfc06b28
          [ 672000 ns] Test is running, debug_wb_pc = 0xbfc07ac8
----[ 676065 ns] Number 8'd35 Functional Test Point PASS!!!
          [ 682000 ns] Test is running, debug_wb_pc = 0xbfc5bc14
          [ 692000 ns] Test is running, debug_wb_pc = 0xbfc5cbb4
----[ 698445 ns] Number 8'd36 Functional Test Point PASS!!!
          [ 702000 ns] Test is running, debug_wb_pc = 0xbfc565fc
          [ 712000 ns] Test is running, debug_wb_pc = 0xbfc57bfc
          [ 722000 ns] Test is running, debug_wb_pc = 0xbfc59250
          [ 732000 ns] Test is running, debug_wb_pc = 0xbfc5a8d4
----[ 736645 ns] Number 8'd37 Functional Test Point PASS!!!
          [ 742000 ns] Test is running, debug_wb_pc = 0xbfc1eaec
          [ 752000 ns] Test is running, debug_wb_pc = 0xbfc20170
          [ 762000 ns] Test is running, debug_wb_pc = 0xbfc217e4
          [ 772000 ns] Test is running, debug_wb_pc = 0xbfc22e94
----[ 774605 ns] Number 8'd38 Functional Test Point PASS!!!
          [ 782000 ns] Test is running, debug_wb_pc = 0xbfc718dc
          [ 792000 ns] Test is running, debug_wb_pc = 0xbfc72f24
          [ 802000 ns] Test is running, debug_wb_pc = 0xbfc7456c
          [ 812000 ns] Test is running, debug_wb_pc = 0xbfc75be4
----[ 812805 ns] Number 8'd39 Functional Test Point PASS!!!
          [ 822000 ns] Test is running, debug_wb_pc = 0xbfc53414
          [ 832000 ns] Test is running, debug_wb_pc = 0xbfc548ec
          [ 842000 ns] Test is running, debug_wb_pc = 0xbfc55d4c
----[ 842485 ns] Number 8'd40 Functional Test Point PASS!!!
          [ 852000 ns] Test is running, debug_wb_pc = 0xbfc29d0c
          [ 862000 ns] Test is running, debug_wb_pc = 0xbfc2b0e0
```

```
          [ 872000 ns] Test is running, debug_wb_pc = 0xbfc2c480
          [ 882000 ns] Test is running, debug_wb_pc = 0xbfc2d884
----[ 886735 ns] Number 8'd41 Functional Test Point PASS!!!
          [ 892000 ns] Test is running, debug_wb_pc = 0xbfc18e80
          [ 902000 ns] Test is running, debug_wb_pc = 0xbfc1a2a0
          [ 912000 ns] Test is running, debug_wb_pc = 0xbfc1b658
          [ 922000 ns] Test is running, debug_wb_pc = 0xbfc1c9f4
          [ 932000 ns] Test is running, debug_wb_pc = 0xbfc1ddcc
----[ 933015 ns] Number 8'd42 Functional Test Point PASS!!!
          [ 942000 ns] Test is running, debug_wb_pc = 0xbfc121b8
          [ 952000 ns] Test is running, debug_wb_pc = 0xbfc13318
          [ 962000 ns] Test is running, debug_wb_pc = 0xbfc14478
          [ 972000 ns] Test is running, debug_wb_pc = 0xbfc155d8
          [ 982000 ns] Test is running, debug_wb_pc = 0xbfc16728
----[ 983765 ns] Number 8'd43 Functional Test Point PASS!!!
          [ 992000 ns] Test is running, debug_wb_pc = 0x00000000
          [1002000 ns] Test is running, debug_wb_pc = 0x00000000
          [1012000 ns] Test is running, debug_wb_pc = 0xbfc7e440
          [1022000 ns] Test is running, debug_wb_pc = 0x00000000
          [1032000 ns] Test is running, debug_wb_pc = 0xbfc7ed14
          [1042000 ns] Test is running, debug_wb_pc = 0x00000000
----[1043835 ns] Number 8'd44 Functional Test Point PASS!!!
          [1052000 ns] Test is running, debug_wb_pc = 0xbfc0ef50
          [1062000 ns] Test is running, debug_wb_pc = 0x00000000
          [1072000 ns] Test is running, debug_wb_pc = 0xbfc0f824
          [1082000 ns] Test is running, debug_wb_pc = 0x00000000
          [1092000 ns] Test is running, debug_wb_pc = 0x00000000
          [1102000 ns] Test is running, debug_wb_pc = 0x00000000
          [1112000 ns] Test is running, debug_wb_pc = 0x00000000
          [1122000 ns] Test is running, debug_wb_pc = 0xbfc10e7c
----[1131265 ns] Number 8'd45 Functional Test Point PASS!!!
          [1132000 ns] Test is running, debug_wb_pc = 0xbfc349e4
          [1142000 ns] Test is running, debug_wb_pc = 0xbfc35984
          [1152000 ns] Test is running, debug_wb_pc = 0xbfc36924
----[1154805 ns] Number 8'd46 Functional Test Point PASS!!!
          [1162000 ns] Test is running, debug_wb_pc = 0xbfc8240c
          [1172000 ns] Test is running, debug_wb_pc = 0xbfc833ac
----[1176105 ns] Number 8'd47 Functional Test Point PASS!!!
          [1182000 ns] Test is running, debug_wb_pc = 0xbfc88c94
----[1185635 ns] Number 8'd48 Functional Test Point PASS!!!
          [1192000 ns] Test is running, debug_wb_pc = 0xbfc17f80
----[1195145 ns] Number 8'd49 Functional Test Point PASS!!!
          [1202000 ns] Test is running, debug_wb_pc = 0xbfc60c04
----[1204035 ns] Number 8'd50 Functional Test Point PASS!!!
```

```
----[1209165 ns] Number 8'd51 Functional Test Point PASS!!!
          [1212000 ns] Test is running, debug_wb_pc = 0xbfc0337c
----[1215135 ns] Number 8'd52 Functional Test Point PASS!!!
----[1221745 ns] Number 8'd53 Functional Test Point PASS!!!
          [1222000 ns] Test is running, debug_wb_pc = 0xbfc00d94
----[1228035 ns] Number 8'd54 Functional Test Point PASS!!!
          [1232000 ns] Test is running, debug_wb_pc = 0x00000000
----[1234645 ns] Number 8'd55 Functional Test Point PASS!!!
----[1241885 ns] Number 8'd56 Functional Test Point PASS!!!
          [1242000 ns] Test is running, debug_wb_pc = 0xbfc00d60
----[1248485 ns] Number 8'd57 Functional Test Point PASS!!!
          [1252000 ns] Test is running, debug_wb_pc = 0xbfc4fa0c
----[1253335 ns] Number 8'd58 Functional Test Point PASS!!!
          [1262000 ns] Test is running, debug_wb_pc = 0xbfc37e08
          [1272000 ns] Test is running, debug_wb_pc = 0xbfc38d3c
----[1277255 ns] Number 8'd59 Functional Test Point PASS!!!
          [1282000 ns] Test is running, debug_wb_pc = 0xbfc67fc4
          [1292000 ns] Test is running, debug_wb_pc = 0xbfc68ee0
          [1302000 ns] Test is running, debug_wb_pc = 0xbfc69e00
----[1302835 ns] Number 8'd60 Functional Test Point PASS!!!
          [1312000 ns] Test is running, debug_wb_pc = 0xbfc2ef90
----[1321725 ns] Number 8'd61 Functional Test Point PASS!!!
          [1322000 ns] Test is running, debug_wb_pc = 0xbfc00ae8
          [1332000 ns] Test is running, debug_wb_pc = 0xbfc4b900
          [1342000 ns] Test is running, debug_wb_pc = 0xbfc4c7e8
----[1342985 ns] Number 8'd62 Functional Test Point PASS!!!
          [1352000 ns] Test is running, debug_wb_pc = 0xbfc44da4
          [1362000 ns] Test is running, debug_wb_pc = 0xbfc45cb4
----[1371175 ns] Number 8'd63 Functional Test Point PASS!!!
          [1372000 ns] Test is running, debug_wb_pc = 0xbfc2ff28
          [1382000 ns] Test is running, debug_wb_pc = 0xbfc30dd0
          [1392000 ns] Test is running, debug_wb_pc = 0xbfc31cf8
----[1397715 ns] Number 8'd64 Functional Test Point PASS!!!
------------------------------------------------------------
[1398137 ns] Error!!!
    reference: PC = 0xbfc00380, wb_rf_wnum = 0x1a, wb_rf_wdata = 0x00004000
    mycpu      : PC = 0xbfc6a074, wb_rf_wnum = 0x09, wb_rf_wdata = 0x41000000
------------------------------------------------------------
------------------------------------------------------------
[1398155 ns] Error(    0)!!! Occurred in number 8'd65 Functional Test Point!
```

4.2 结果分析与讨论

从日志输出中可以看到，在大多数测试点（如 8’d01、8’d02、8’ d60 等），测试都顺利通过，功能测试点标记为 PASS，且没有错误信息。每个测试点后跟着的是 debug_wb_pc 的值，表示当前测试的程序计数器值。这些信息意味着在每个测试点，CPU 的工作是正常的，且没有异常发生。且在调试过程中，debug_wb_pc 的值逐渐变化，符合预期。

5. 实验总结与展望

5.1 王新宇

这次实验是我第一次深入接触计算机体系结构中的流水线设计，通过参与《计算机系统》课程实验，我不仅对五级流水线有了更深刻的理解，还在团队合作和问题解决方面获得了宝贵的经验。

IF 模块：在实现 IF 模块时，我深刻理解了指令取指阶段的重要性。通过控制指令延迟槽和跳转指令，确保了指令的正确取指和执行顺序。这一部分让我认识到流水线控制的复杂性，尤其是在处理跳转指令时，如何正确管理延迟槽是一个值得深思的问题。**ID 模块**：ID 模块的实现让我对指令译码和数据相关性有了更深入的了解。通过处理寄存器读写和数据相关性，我学会了如何在流水线中进行有效的资源管理，避免数据冲突。**MEM 和 WB 模块**：在实现 MEM 和 WB 模块时，我体会到了内存访问和结果写回的复杂性。确保 load 和 store 指令的正确执行，以及结果正确写回寄存器堆，是保证程序正确性的关键。**STALL 控制**：在实现 STALL 控制时，我深刻理解了流水线停顿的机制。通过控制流水线停顿，避免了数据冲突，保证了指令的正确执行。

在这次实验中，我深刻体会到了团队合作的重要性。我们小组分工明确，各司其职，但又相互协作，共同解决实验中遇到的问题。在调试过程中，我们经常一起讨论问题的原因，共同寻找解决方案，这种团队合作的精神让我受益匪浅。

5.2 刘力瑞

在过去一段时间里，我通过学习和实践，深入了解了计算机体系结构，特别是关于 EX 架构的指令执行过程、硬件设计中的控制信号生成以及 Verilog 硬件描述语言的使用。通过这些学习，我对计算机硬件和软件之间的关系、如何通过硬件设计实现指令集的功能有了更深刻的理解。在架构中，每一条指令都通过多个阶段处理，包括取指、解码、执行、访存和

写回等阶段，而我负责的阶段主要是 EX 执行操作。

EX 模块承担着执行运算、计算地址和 ALU 结果的关键任务。通过参与其中，我对各类算术和逻辑运算的底层实现有了更为透彻的理解。从简单的加减法到复杂的乘除法，不仅明白了其数学原理在硬件层面的转化方式，更掌握了如何通过控制信号精确地指挥 ALU 完成这些运算。

这些硬件操作通过控制信号来协调完成，因此如何生成这些控制信号成为了硬件设计的关键。在学习硬件描述语言 Verilog 的过程中，我掌握了如何用代码描述硬件模块、如何实现寄存器、ALU、乘法器、除法器等基本模块的设计。

例如，在实现乘法和除法运算时，我学到了如何设计乘法器和除法器，并通过控制信号判断是否需要开始运算、是否需要等待结果。通过设计乘法和除法运算的流水线，我加深了对硬件逻辑的理解，还学会了如何通过代码模拟这些硬件模块的行为。

在学习如何生成控制信号时，我意识到它们在硬件中的重要性。例如，hi_we 和 lo_we 信号控制了是否将乘法和除法的结果写入 HI 和 LO 寄存器，

6 参考资料

1、张晨曦 著《计算机体系结构》（第二版） 高等教育出版社
2、雷思磊 著《自己动手写 CPU》 电子工业出版社