实验报告成绩:	成绩评定日期:

2025 ~ 2026 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 2301

组长：周炜杰

组员：李天牧

报告日期：2025.12.30

1. 实验设计

1.1 小组成员工作量划分

姓名	工作内容	工作量占比
周炜杰	添加部分算数指令、逻辑运算指令、数据移动指令、跳转指令，访存指令，数据前推，参与实现 stall，参与实验报告编写	60%
李天牧	实现 hilo 寄存器，算数指令，乘除法指令，参与实现 stall，参与实验报告编写	40%

1.2 运行环境及使用工具

运行环境：本地 Vivado 19.2。

编程环境：VSCode 编写代码，Vivado 模拟仿真，git 版本管理，Github 搭建项目仓库。

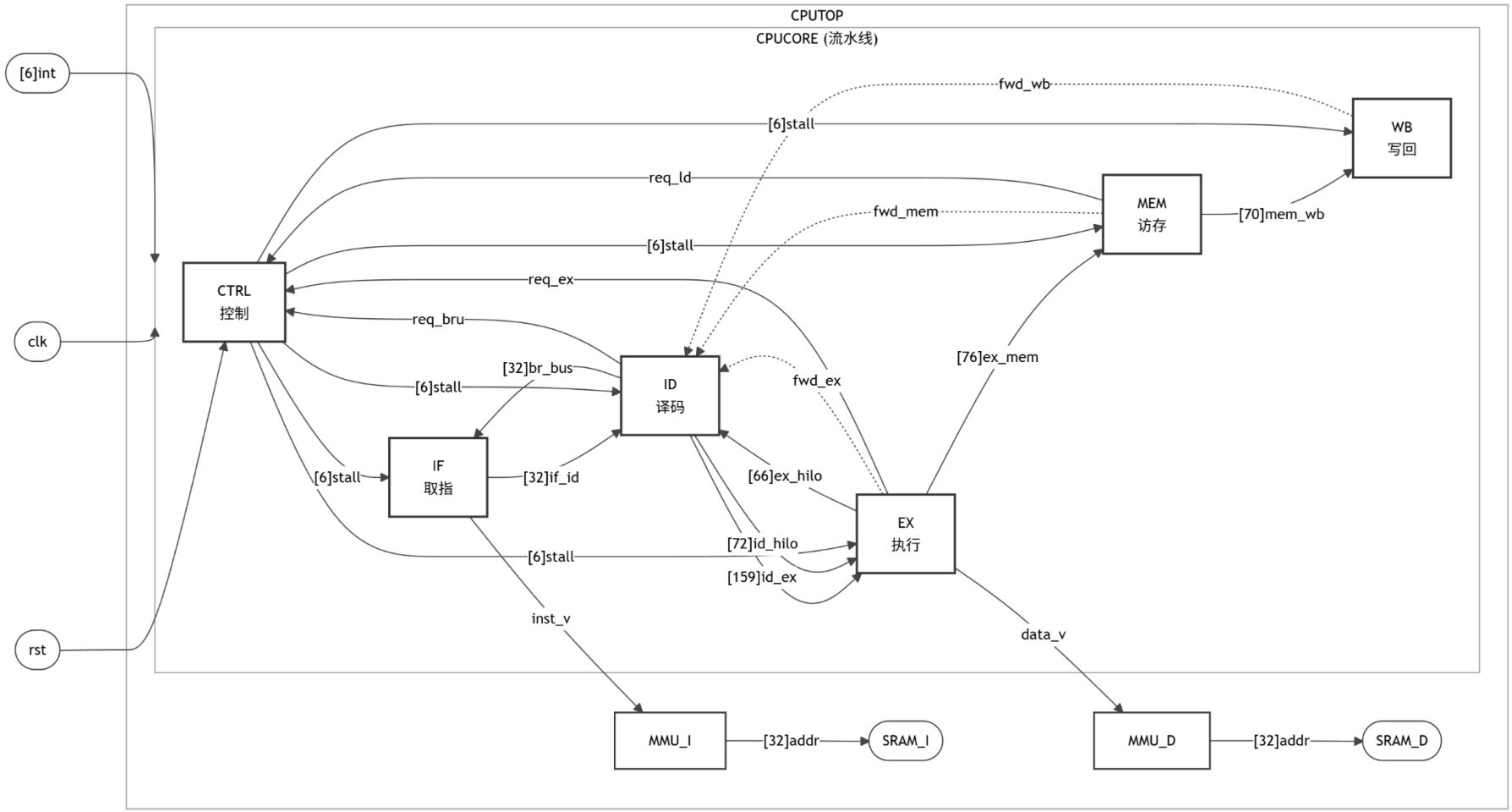
1.3 总体设计

项目包括 IF.v，ID.v，EX.v，MEM.v，WB.v，hi_lo_reg.v，mycpu_core.v， mycpu_top.v，搭建了一条流水线的基本框架。

本次实验实现了一个基于 MIPS32 指令集架构的 5 级静态流水线 CPU。包括取指（IF）、译码（ID）、执行（EX）、访存（MEM）和写回（WB）五级流水线。

为解决 RAW（Read After Write）数据冒险，设计了前推（Forwarding）机制。ID 端接收来自 EX、MEM 和 WB 阶段的新数据，逻辑优先级为：EX 前推 > MEM 前推 > WB 前推 > 寄存器堆读取。

为解决访存冲突，多周期除法运算暂停问题，引入流水线暂停，由 CTRL 模块控制。采用 ID 阶段跳转。在 ID 阶段直接计算比较结果和跳转地址，通过 br_bus 反馈给 IF 阶段。


CPUTOP
CPUCORE (流水线)
[6]int
clk
rst
CTRL
控制
IF
取指
ID
译码
EX
执行
MEM
访存
WB
写回
MMU_I
MMU_D
SRAM_I
SRAM_D
fwd_wb
[6]stall
req_ld
fwd_mem
[70]mem_wb
[6]stall
req_ex
req_bru
[32]br_bus
[6]stall
fwd_ex
[76]ex_mem
[6]stall
[32]if_id
[66]ex_hilo
[6]stall
[72]id_hilo
[159]id_ex
inst_v
data_v
[32]addr
[32]addr

2. 流水线各阶段的说明

2.1 IF 模块

整体说明：IF（取指）模块是 CPU 流水线的第一级，其核心任务是管理程序计数器（PC）并从指令存储器中获取指令。

具体功能说明：

PC 管理：维护当前指令的地址。在复位时初始化 PC，在正常运行时根据逻辑更新 PC。

指令获取：将 PC 地址发送给指令存储器，并控制片选信号。

下一跳地址计算：计算下一个时钟周期的 PC 值。默认情况下为顺序执行（PC + 4），当接收到 ID 阶段传来的跳转信号时，更新为跳转目标地址。

流水线控制：响应来自 CTRL 模块的暂停信号。当发生流水线暂停时，保持当前 PC 值不变。

端口介绍：

表 IF 模块输入输出

端口名称	方向	位宽	描述
clk	Input	1	时钟信号
rst	Input	1	复位信号
stall	Input	6	流水线暂停信号总线
br_bus	Input	33	来自 ID 阶段的分支跳转总线，包含跳转使能和目标地址
if_to_id_bus	Output	33	发往 ID 阶段的数据总线，包含当前 PC 值和指令有效位
inst_sram_en	Output	1	指令存储器读使能信号
inst_sram_wen	Output	4	指令存储器写使能信号
inst_sram_addr	Output	32	发往指令存储器的地址（PC）
inst_sram_wdata	Output	32	发往指令存储器的写数据

信号介绍：

pc_reg (Reg): 程序计数器寄存器，时序逻辑，存储当前的指令地址。

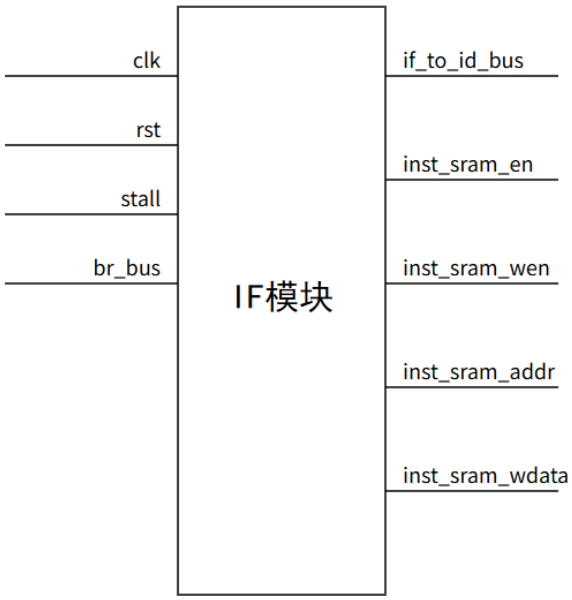
ce_reg (Reg): 片选寄存器，指示当前 CPU 是否处于工作状态。

next_pc (Wire): 组合逻辑计算出的下一条指令地址。

br_e (Wire): 分支跳转使能信号。

br_addr (Wire): 跳转目标地址。

结构示意图：



2.2 ID 模块

整体说明：ID（译码）负责将 IF 阶段获取的二进制指令翻译成具体的操作信号。对指令进行译码，然后将结果传给 EX 段，实现寄存器的读写，处理数据。

具体功能说明：

指令译码：将 32 位指令拆解，识别操作码（Opcode）和功能码（Funct），判断当前指令类型。依据指令中的特征字段区分指令，同时激活相应的指令对应的 inst_**变量，表示是哪一条指令。根据译码结果，读取通过 regfile 模块读取 地址为 rs（inst[25;21]）以及地址为 rt(inst[20:16]的通用寄存器,得到 rdata1 以及 rdata2，并且通过判断是否发生数据相关，从而更改 rdata1 以及 rdata2 的值。

操作数读取与前推：从 RegFile 读取源操作数。实现数据前推逻辑，能从 EX、MEM、WB 阶段直接获取最新的寄存器数据，解决 RAW 冒险。

分支跳转判断：在译码阶段直接比较操作数，判断分支是否成立并计算跳转目标地址，将结果通过 br_bus 立即反馈给 IF 阶段。

流水线控制信号生成：生成 ALU 的控制码、访存使能信号、寄存器写使能信号等，并打包发送给 EX 阶段。

冒险检测：检测 Load-Use 冒险，当上一条指令是 Load 且发生数据冲突时，发出 stallreq 请求暂停流水线。

端口介绍：

表 ID 模块输入输出

端口名称	方向	位宽	描述
clk	Input	1	时钟信号
rst	Input	1	复位信号
stall	Input	6	流水线暂停信号
if_to_id_bus	Input	33	来自 IF 阶段的数据（PC 值）
inst_sram_rdata	Input	32	指令存储器返回的指令数据

wb_to_rf_bus	Input	38	来自 WB 阶段的数据（用于写回 RegFile 及前推）
ex_to_rf_bus	Input	38	来自 EX 阶段的前推数据
mem_to_rf_bus	Input	38	来自 MEM 阶段的前推数据
id_to_ex_bus	Output	159	发往 EX 阶段的解码信息包（操作数、ALU 控制码等）
br_bus	Output	33	发往 IF 阶段的分支跳转总线
stallreq_for_bru	Output	1	发往 CTRL 的暂停请求（检测到 Load-Use 冒险）

信号介绍：

inst：当前正在译码的 32 位指令。

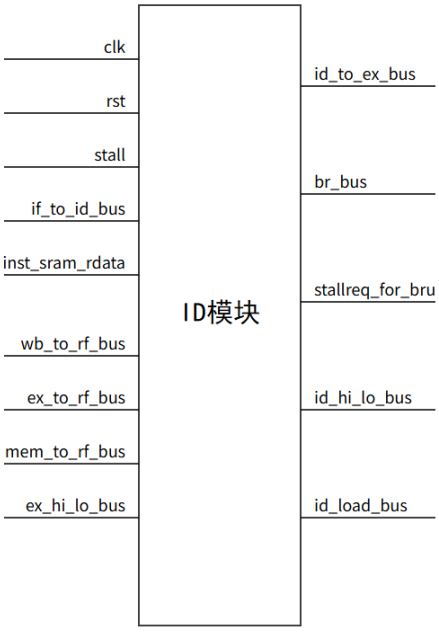
rs，rt，rd：从指令中提取的源寄存器和目标寄存器地址。

rdata1，rdata2：从寄存器堆直接读出的原始数据。

ndata1，ndata2：经过前推选择器处理后的最终操作数，这是 ALU 实际使用的数据。

alu_op：译码生成的 ALU 操作码，指示 ALU 进行何种运算。

结构示意图：



2.3 EX 模块

整体说明：

这部分负责执行运算、地址计算以及计算 ALU 的结果。从 ID/EX 流水线的寄存器中读取由**寄存器 1** 传来的值和**寄存器 2** 传来的值（或**寄存器 1** 传来的值和符号扩展过后的立即数的值），并用 ALU 将它们相加，结果值存入 EX/MEM 流水线寄存器。（这里 ALU 模块是提供的）

具体功能说明：

ALU 运算：根据 ALU 操作码（alu_op），对源操作数进行加减、逻辑运算（与或非异或）、移位等操作。

乘除法运算：内置独立的乘法器和除法器模块。特别是除法运算，由于需要多个时钟周期，该模块负责向 CTRL 发出暂停请求以冻结流水线，直到除法完成。

访存地址计算与数据对齐：对于 Load/Store 指令，EX 阶段通过 ALU 计算内存物理地址。对于 Store 指令，还需要根据地址的低两位处理写入数据的字节对齐和复制。

HI/LO 寄存器处理：处理乘除法结果的写入以及 MFHI/MTHI 等指令的数据传递。

数据前推源生成：将计算结果（ex_result）打包发送回 ID 阶段（ex_to_rf_bus），作为数据前推的最优先来源。

端口介绍：

表 EX 模块的输入输出

端口名称	方向	位宽	描述
clk	Input	1	时钟信号
rst	Input	1	复位信号
stall	Input	6	流水线暂停信号

id_to_ex_bus	Input	159	来自 ID 阶段的解码信息包（操作数、立即数、控制信号等）
id_hi_lo_bus	Input	72	来自 ID 阶段的 HI/LO 寄存器相关信号
ex_to_mem_bus	Output	76	发往 MEM 阶段的数据包（运算结果、写回地址等）
ex_to_rf_bus	Output	38	发往 ID 阶段的前推数据包
data_sram_en	Output	1	数据存储器使能信号
data_sram_wen	Output	4	数据存储器字节写使能信号
data_sram_addr	Output	32	数据存储器地址
data_sram_wdata	Output	32	数据存储器写数据
stallreq_for_ex	Output	1	发往 CTRL 的暂停请求（用于多周期除法）

信号介绍：

alu_op: ALU 操作控制码，决定 ALU 执行何种运算。

alu_src1, alu_src2: 经过选择器选择后的 ALU 最终源操作数。

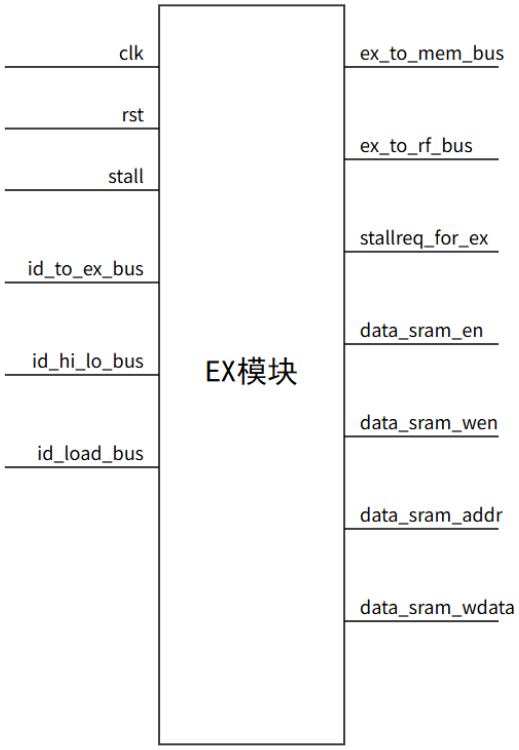
alu_result: ALU 的运算结果。

mul_result, div_result: 乘法和除法模块的运算结果。

byte_sel: 字节选择信号，根据地址的低两位生成，用于非对齐访存。

data_ram_sel: 最终发往数据存储器的字节写掩码。

结构示意图：



2.4 MEM 模块

整体说明：

执行访问内存操作，从 EX、MEM 流水线寄存器中得到地址读取数据寄存器，并将得到的数据存入到 MEM、WB 的流水线寄存器中，接收并处理访存的结果，并对结果进行选择写回。

具体功能说明：

接收 EX 阶段传来的 ALU 运算结果或访存地址，读取内存，接收从数据存储器返回的原始 32 位数据（data_sram_rdata)。由于 MIPS 的 LB 和 LH 指令需要从 32 位字中提取特定部分，根据字节选择信号（data_ram_sel）进行数据截取。根据具体的 Load 指令类型，对截取的数据进行零扩展或符号扩展，生成最终的 32 位加载数据。根据指令类型选择写回寄存器的数据来源。将最终结果（rf_wdata）打包发送回 ID 阶段（mem_to_rf_bus)，解决 RAW 数据冒险。

端口介绍：

表 MEM 模块输入输出

端口名称	方向	位宽	描述
clk	Input	1	时钟信号。
rst	Input	1	复位信号。
stall	Input	6	流水线暂停信号。
ex_to_mem_bus	Input	76	来自 EX 阶段的数据包。
data_sram_rdata	Input	32	从数据存储器读回的原始 32 位数据。
data_ram_sel	Input	4	来自 EX 阶段的字节有效掩码，指示数据在 32 位中的位置。
ex_load_bus	Input	5	来自 EX 阶段的 Load 指令类型标志。
mem_to_wb_bus	Output	70	发往 WB 阶段的数据包。
mem_to_rf_bus	Output	38	发往 ID 阶段的前推数据包。
stallreq_for_load	Output	1	发往 CTRL 的暂停请求。

信号介绍：

ex_result: 透传自 EX 阶段的 ALU 计算结果。

b_data: 从 32 位内存数据中提取出的 8 位字节数据。

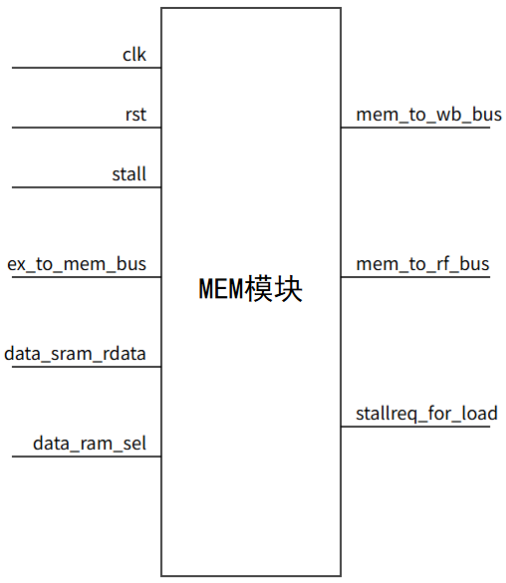
h_data: 从 32 位内存数据中提取出的 16 位半字数据。

mem_result: 经过扩展处理后的内存数据。

rf_wdata: 最终决定写入寄存器堆的数据。

sel_rf_res: 结果选择信号，1 代表选择内存结果，0 代表选择 ALU 结果。

结构示意图：



2.5 WB 模块

整体说明：

将结果写回寄存器，从 MEM、WB 流水线寄存器中读取数据并将其写回中部寄存器堆中。

具体功能说明：

该模块中有 9 个输入输出的端口。包括时钟（clk）、复位（rst）、stall，mem_to_wb_bus，以及调试信号。

输出包括 wb_to_rf_bus，以及 debug 用的调试信号。

将 MEM 阶段传递过来的最终执行结果（rf_wdata）写入通用寄存器堆（RegFile）。通过 wb_to_rf_bus 总线将写使能、写地址和写数据回传给 ID 阶段，数据前推，供后续指令在 ID 阶段直接使用。

端口介绍：

表 WB 模块的输入输出

端口名称	方向	位宽	描述
clk, rst	Input	1	时钟与复位信号。
stall	Input	6	流水线暂停信号。
mem_to_wb_bus	Input	70	来自 MEM 阶段的数据包（PC、写使能、写地址、写数据）。
wb_to_rf_bus	Output	38	关键输出：发回 ID 阶段的总线，连接寄存器堆的写端口。
debug_wb_pc	Output	32	调试信号：当前提交指令的 PC。
debug_wb_rf_wen	Output	4	调试信号：寄存器组写使能。
debug_wb_rf_wnum	Output	5	调试信号：写入的目标寄存器号。
debug_wb_rf_wdata	Output	32	调试信号：写入的寄存器数据。

信号介绍：

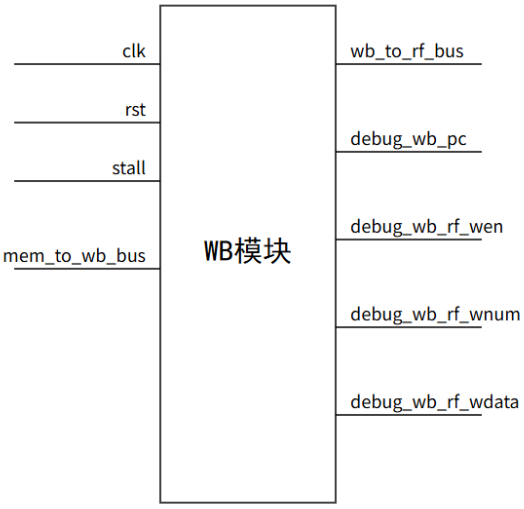
wb_pc：当前指令的程序计数器值，用于调试记录。

rf_we：寄存器写使能信号。

rf_waddr：目标寄存器地址。

rf_wdata：最终写入寄存器的数据。

结构示意图：



2.6 CTRL 模块

整体说明：

协调流水线。控制流水线暂停。

端口介绍：

表 CTRL 模块输入输出

端口名称	方向	位宽	描述
rst	Input	1	复位信号。
stallreq_for_ex	Input	1	来自 EX 阶段的请求（如除法器忙）。
stallreq_for_bru	Input	1	来自 ID 阶段的请求（如 Load-Use 冒险）。
stall	Output	6	全局暂停总线：发往所有流水线阶段。

信号介绍：

stall[0]：控制 PC 指针是否更新。

stall[1]：控制 IF/ID 流水线寄存器。

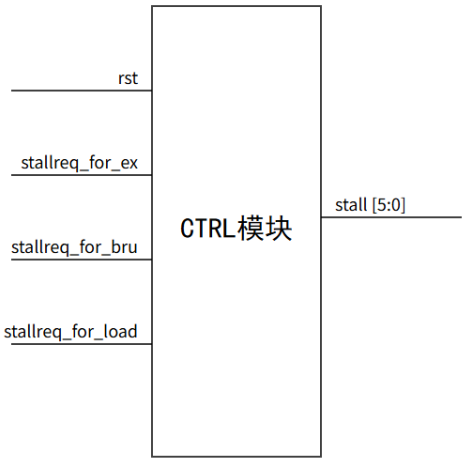
stall[2]：控制 ID/EX 流水线寄存器。

stall[3]：控制 EX/MEM 流水线寄存器。

stall[4]：控制 MEM/WB 流水线寄存器。

stall[5]：控制 WB 阶段。

结构示意图：



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。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。

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

端口介绍：

表 HILO 寄存器输入输出

端口名称	方向	位宽	描述
clk	Input	1	时钟信号。
Stall	Input	6	暂停信号。
hi_we	Input	1	HI 寄存器写使能信号。
lo_we	Input	1	LO 寄存器写使能信号。
hi_wdata	Input	32	待写入 HI 寄存器的数据。
lo_wdata	Input	32	待写入 LO 寄存器的数据。
hi_rdata	Output	32	HI 寄存器的当前值（读出）。
lo_rdata	Output	32	LO 寄存器的当前值（读出）。

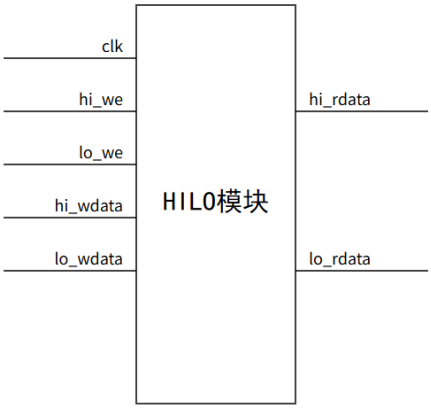
信号介绍：

HI (Reg): 高位寄存器，复位时清零。

LO (Reg): 低位寄存器，复位时清零。

hi/lo_we: 写使能控制，高电平有效。当该信号有效时，在时钟上升沿将 wdata 写入对应的寄存器。

结构示意图：



3. 实验感受及意见

3.1 周炜杰部分

在本次实验中，我深入学习并实践了基于 Verilog 和 MIPS 指令集的 CPU 流水线涉及。通过实操感受流水线各阶段的功能，数据传递和控制。同时，对数据冒险，分支跳转等流水线冲突有了进一步的认识。

通过本次实验我加深了对 Git 的认识，之前对 Git 的使用大都是一次性的上传下载，而忽视了其强大的版本管理功能，而这也是 Git 的初衷。同时，我熟练了对 Github 的使用，掌握了搭建仓库、管理版本以及 Git 分支与合并等基本操作，提升了工作效率。

对于实验的建议。希望在实验课上加入 Verilog 的入门等具体教程，以及 debug 方法的演示，纯自学以及整理资料和任务花费了大量不必要的时间。

3.2 李天牧部分

实验初期，我秉持准工程师的系统思维，通过 Git 实践代码的可追溯开发，并利用 Vivado 平台将调试过程从盲目尝试转向以结果为导向的迭代跃迁。在任务分工中，我负责最具挑战性的 HI/LO 寄存器模块及乘除法指令链设计（Point 44-58）。针对除法器接入带来的多周期延时，我设计并验证了流水线暂停（Stall）机制，有效解决了执行顺序失稳的难题。当乘除法指令最终精准通过压力测试，我不仅攻克了打破传统“三地址”模式的技术跨度，更在实践中对硬件设计中“时序平衡”与“空间换时间”的平衡艺术有了入木三分的理解。

回望整段实验历程，从最初面对空荡代码框架时的举棋不定，到最后看着 Vivado 输出结果中那一行行代表成功的“Pass”标志，这是一场从理论知识到底层逻辑的完整重塑。我不仅亲手实现了一套涵盖复杂计算逻辑的处理机制，更通过这种“剥洋葱”式的实践，建立起了对计算机系统的全栈式认知。我开始明白，在软件层面的每一次运算背后，底层硬件正经历着极其精密的数据搬运、时序校验与冲突消解。这种透视底层的能力，磨砺了我面对未知挑战时的沉着与定力。实验虽然告一段落，但这种严谨的工程师素养、高效的学习策略以及对系统底层永不熄灭的探索热情，将成为我职业生涯中最宝贵的财富。

4．参考资料

1．龙芯杯参赛资料

2．张晨曦《计算机体系结构》

3．雷思磊《自己动手写 CPU》