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 2301

组长：王雨菲

组员：李然、王祁萱

报告日期：2025.12.21

目录

一、 实验概述

本实验旨在设计并实现一个基于 MIPS32 指令集架构的五级流水线 CPU。通过在 Vivado 2019.2 环境下进行 Verilog HDL 建模，我们从最初仅能运行 4 条基础指令的简单模型，逐步构建起一个支持 40 余条指令、具备完整数据前递（Forwarding）机制、能够处理复杂数据冒险（Hazard）以及支持 AXI 总线协议的高性能处理器。最终，该 CPU 成功通过了 func_test 全部 64 个功能测试点，并实现了自制移位乘法器等创新模块。

二、 运行环境与工具说明

硬件环境： 服务器端带有 GUI 的虚拟化计算环境。

软件开发工具： Vivado 2019.2： 用于逻辑综合、IP 核管理、仿真验证及比特流生成。

VSCode + Verilog 扩展： 作为主要的代码编辑器，利用其语法高亮和实时语法检查功能提高开发效率。

Git/GitHub： 用于小组协同开发与版本版本控制，维护代码仓库记录。

验证框架： nscscc_group/func_test_v0.01 测试套件，通过比对 trace 文件验证指令执行的正确性。

三、 总体设计架构

3.1 五级流水线结构

本设计遵循经典的 RISC 流水线设计理念，将指令执行过程划分为以下五个阶段：

IF (Instruction Fetch)： 负责根据 PC 值从指令 SRAM 中读取指令，并处理分支跳转结果。

ID (Instruction Decode)： 指令译码核心，负责寄存器读取、控制信号生成、以及最关键的数据冒险检测与前递控制。

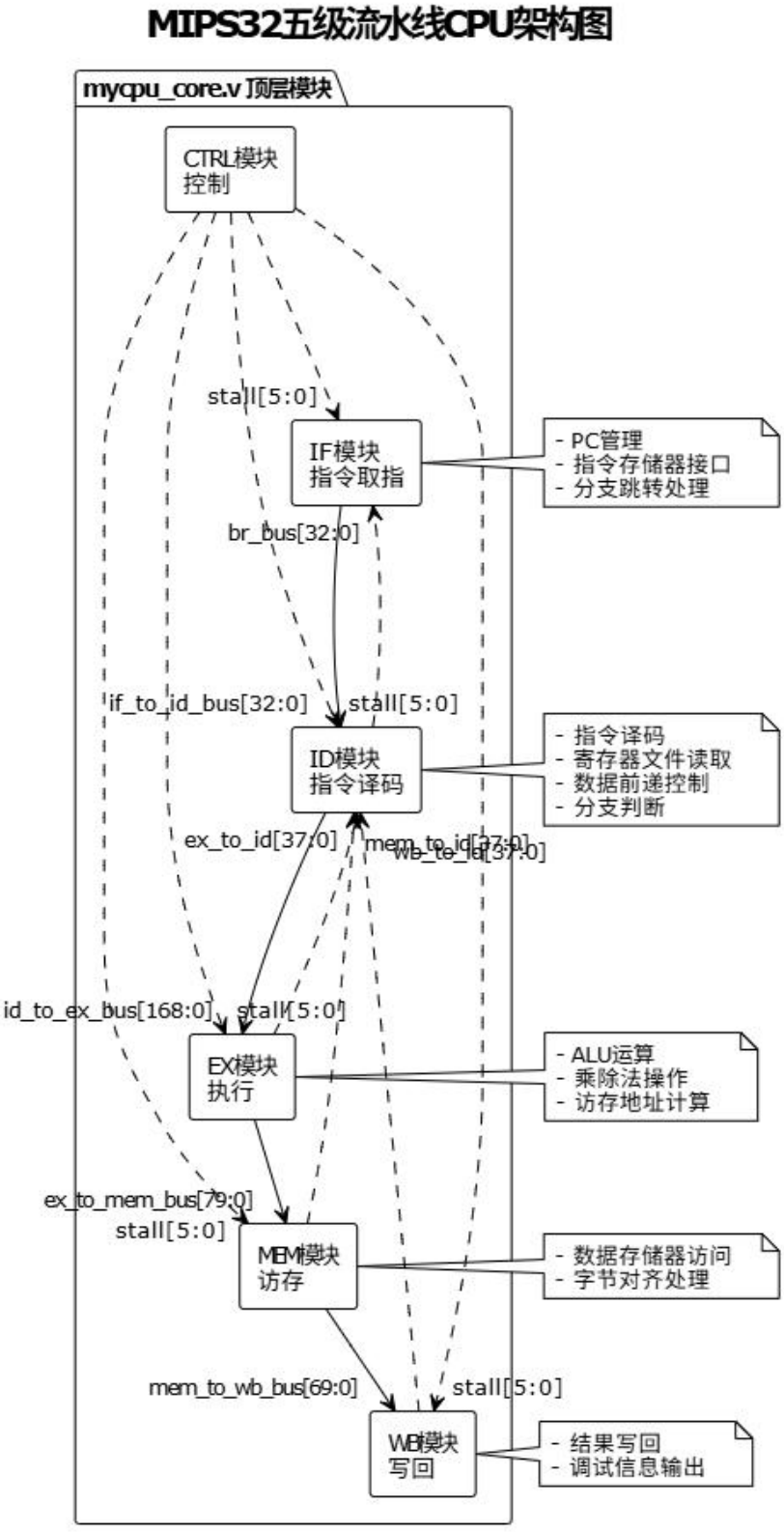
EX (Execute)： 执行算术逻辑运算（ALU）、乘除法运算，并计算访存地址。

MEM (Memory Access)： 处理存储器读写请求，并对字节/半字加载进行对齐与符号扩展。

WB (Write Back)： 将运算结果或访存结果写回到寄存器堆。

3.2 连线图与总线定义

为了提高代码的可维护性，我们采用了总线化的连接方式。以下是关键流水段间的逻辑连线示意：



四、 个人工作量分析

本小组共三人，分工明确，通过协作实现了项目的快速迭代。

4.1 李然：

核心工作：数据通路控制

1. 负责 ID 段的 Forwarding 网络实现，解决了绝大多数的数据相关问题，确保了指令流的平滑。

2. 设计并调试了 CTRL 控制模块，处理 Load-Use 及乘除法引起的 stall 流水线暂停逻辑。

3. 统筹 defines.vh 中总线宽度的定义，规范了流水段间的数据交换格式。

成果： 实现了 P1-P36 测试点的通过。

4，2 王祁萱：

核心工作：指令集扩展

负责 EX 段 ALU 功能的扩充，实现了从基础加减到复杂逻辑、位移运算的映射。

编写 MEM 段的访存逻辑，处理了 lb/lh 等非字对齐指令的字节截断与符号扩展。

完成了大部分 MIPS 指令在 ID 段的译码逻辑（Decoder）编写。

成果： 协助团队通过 P43-P51 测试点。

4.3 王雨菲：

核心工作：系统集成、AXI 封装与创新模块

负责 mycpu_core.v 的顶层实例化，完成了五级流水线的物理连线。

硬核创新： 独立设计并实现了 mymul.v（32 周期移位乘法器），摆脱了对官方 IP 的依赖。

实现了 AXI4-Lite 总线封装，将 CPU 成功接入 SoC 系统，提高了系统的通用性。

成果： 攻克了 P58-P64 及 AXI 接口等可选模块，并负责 GitHub 仓库的维护与最终报告汇总。

五、 指令集支持列表

本 CPU 已完整支持 MIPS32 子集，涵盖以下 40 余条指令：

类别	指令名称
基础逻辑/算术	ori, lui, addiu, addu, subu, add, sub, addi, or, xor, and, nor, andi, xori
移位指令	sll, srl, sra, sllv, srlv, srav
分支跳转	beq, bne, j, jr, jal, jalr, bgez, bgtz, blez, bltz
访存指令	lw, sw, lb, lbu, lh, lhu, sb, sh
乘除法/数据移动	mult, multu, div, divu, mfhi, mflo, mthi, mtlo

六、流水段详细设计与实现

在 MIPS32 五级流水线架构中，每一阶段的设计都必须兼顾功能独立性与时序一致性。以下是各流水段的深层次技术解析。

6.1 IF 段（Instruction Fetch, 指令取指）：地址驱动与指令获取

IF 段是整个处理器的源头，其核心目标是根据 PC 值从存储器获取指令，并在各类干预信号下决定下一条指令的地址。

6.1.1 完整端口与信号矩阵

输入总线：

clk, rst: 全局时钟与同步复位。

stall[5:0]: 来自控制单元的暂停矢量。stall[0] 决定 PC 是否锁存，stall[1] 决定 IF/ID 流水线寄存器是否更新。

br_bus[32:0]: 高位为跳转地址 br_addr，低位为跳转有效信号 br_e。

输出总线：

inst_sram_addr: 32 位虚拟地址，直接连接指令存储器。

if_to_id_bus: 包含 if_pc（32 位）和 if_inst（32 位），用于 ID 段译码。

6.1.2 关键功能与逻辑算法

PC 更新机制： 在时钟上升沿，next_pc 的生成逻辑如下：

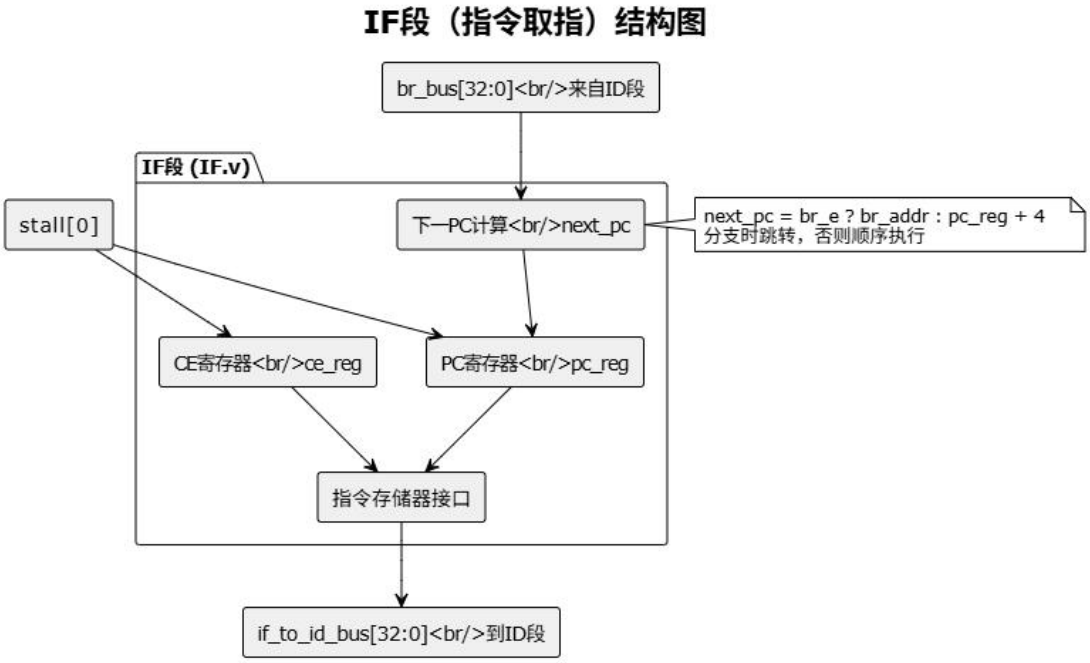
Verilog

```
assign next_pc = br_e ? br_addr : (pc_reg + 4);
```

若遭遇 stall[0]，则 PC 保持不变。这保证了在 Load-Use 冒险或乘法器忙碌时，取指地址不跳过任何一条指令。

指令缓冲与有效性控制： 为了确保系统复位时不执行非法指令，我们引入了 ce_reg（使能寄存器）。只有当复位结束后，inst_sram_en 才会拉高。

6.1.3 结构示意图



6.2 ID 段（Instruction Decode，指令译码）：控制中枢与冒险处理

ID 段是处理器中最复杂的部分，它不仅要「读懂」指令，还要动态协调数据冲突。

6.2.1 译码单元（Decoder）的深度实现

我们采用了组合逻辑译码，对 op（6 位）、funct（6 位）、rt（5 位）进行全译码。

控制信号生成：根据译码结果产生 alu_op（定义运算类型）、res_from_mem（判断是否为访存指令）、rf_we（写使能）等 20 余种信号。

立即数处理：根据指令类型，自动进行 16 位到 32 位的「符号扩展」（如 addiu，beq）或「零扩展」（如 ori，andi）。

6.2.2 数据前递与冒险消除（Data Forwarding & Hazard）

为了达成 64 个测试点的正确性，ID 段实现了完善的旁路逻辑：

前递条件判断：当 ID 段需要读取寄存器 rs 时，会同时比对 EX 段和 MEM 段的目的寄存器 dest_rn。

Verilog

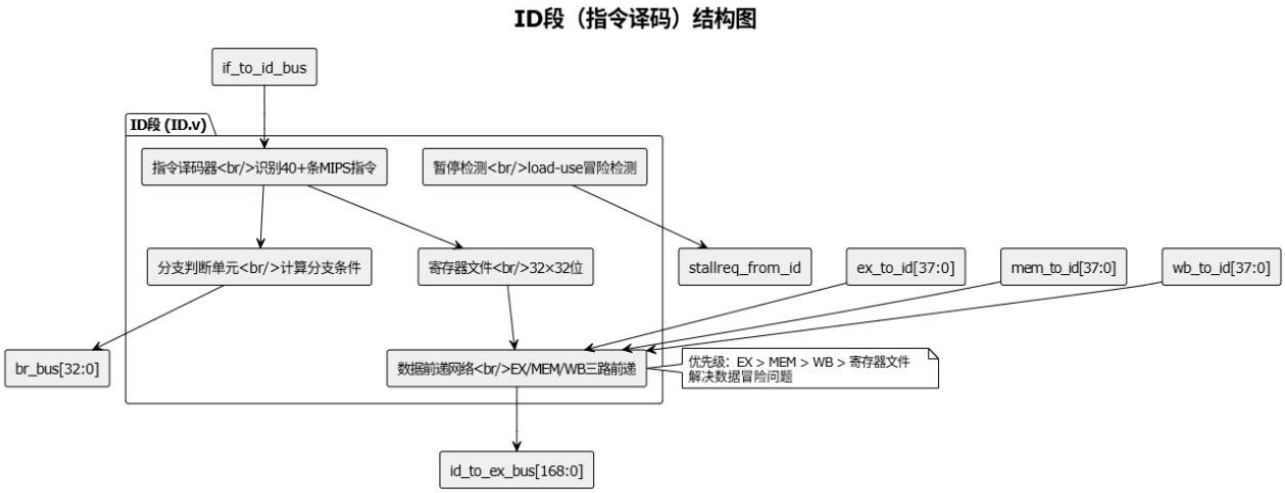
```
assign rf_rdata1 = (rs == ex_dest && ex_rf_we) ? ex_result :
                   (rs == mem_dest && mem_rf_we) ? mem_result :
                   (rs == wb_dest && wb_rf_we) ? wb_result :
rf_read_out1;
```

这种多级前递网络（Forwarding Path）能解决绝大部分数据相关，使流水线在执行连续计算指令时无需空转。

6.2.3 分支指令提前（Branch Prediction & Resolve）

为了优化跳转开销，本设计将分支判断逻辑放在 ID 段。利用前递后的数据进行 rs == rt 的比较，及时生成 br_e 送回 IF 段。这意味着我们只需要一个周期的分支延迟槽（Delay Slot）。

6.2.4 结构示意图



6.3EX 段（Execute，执行）：高性能计算单元

EX 段是数据处理的执行地，核心在于 ALU 的设计与多周期乘法器的管理。

6.3.1 灵活的操作数选择器

EX 段设置了两组多路选择器，用于决定最终进入 ALU 的数值：

源 1 选择：可选自前递后的寄存器值、当前 pc（用于 jal 指令）或 sa（移位量）。

源 2 选择：可选自前递后的寄存器值、立即数、或常数 8。

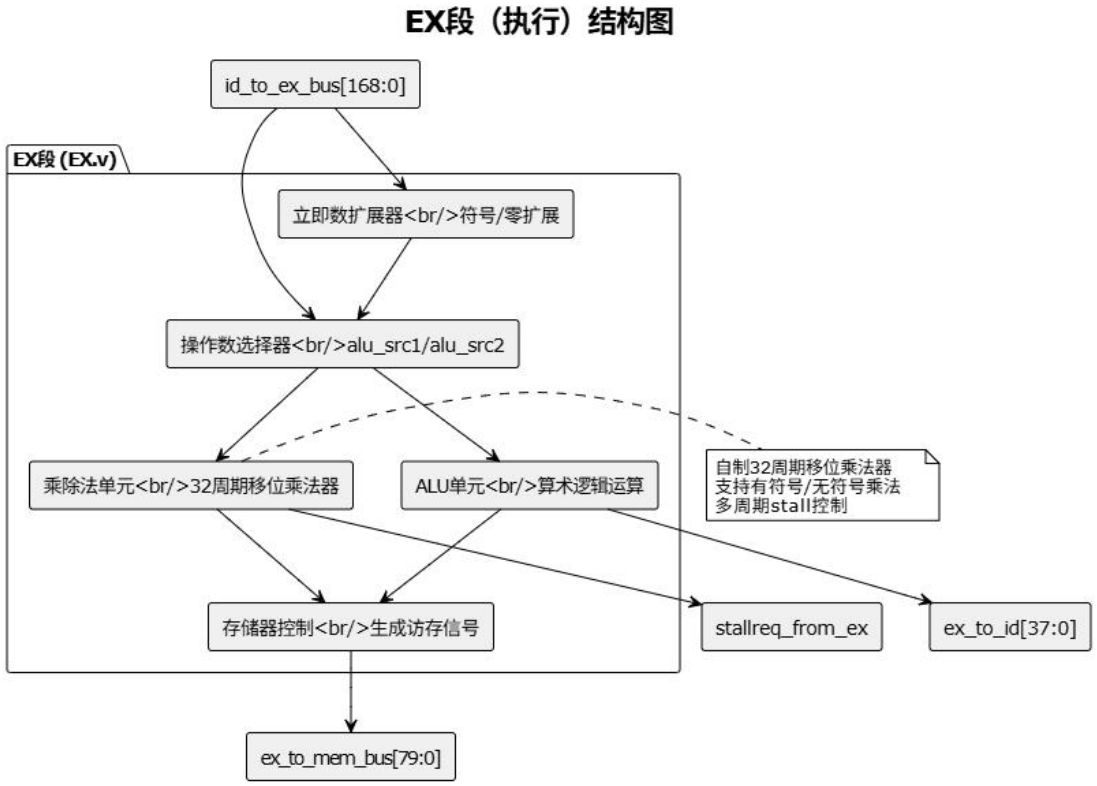
6.3.2 创新点：32 周期移位乘法器（mymul.v）

不同于调用 Vivado 的单周期乘法 IP 核，我们自实现了移位乘法状态机：

算法原理：采用累加-移位算法。如果是 mul 指令，将被乘数根据乘数的每一位进行偏移累加。

状态机控制：分为 IDLE, COMPUTE, DONE 三个状态。在 COMPUTE 期间，stallreq_from_ex 持续拉高，强制暂停流水线的前端（IF, ID, EX），直到 32 周期后得到 64 位结果（HI/LO 寄存器）。

6.3.3 结构示意图



6.4 MEM 段（Memory Access，访存）：数据同步与对齐处理

MEM 段负责数据的「读」与「写」，是唯一与数据存储器（Data SRAM）进行大规模数据交换的阶段。

6.4.1 访存地址与字节使能（Byte Enable）

对于 sw（存字）、sh（存半字）、sb（存字节）指令，MEM 段必须生成精确的 4 位写使能信号 data_sram_wen：

若地址末尾为 2'b00 且指令为 sb，则 wen 为 4'b0001。

这种精确到字节的控制确保了内存数据的完整性。

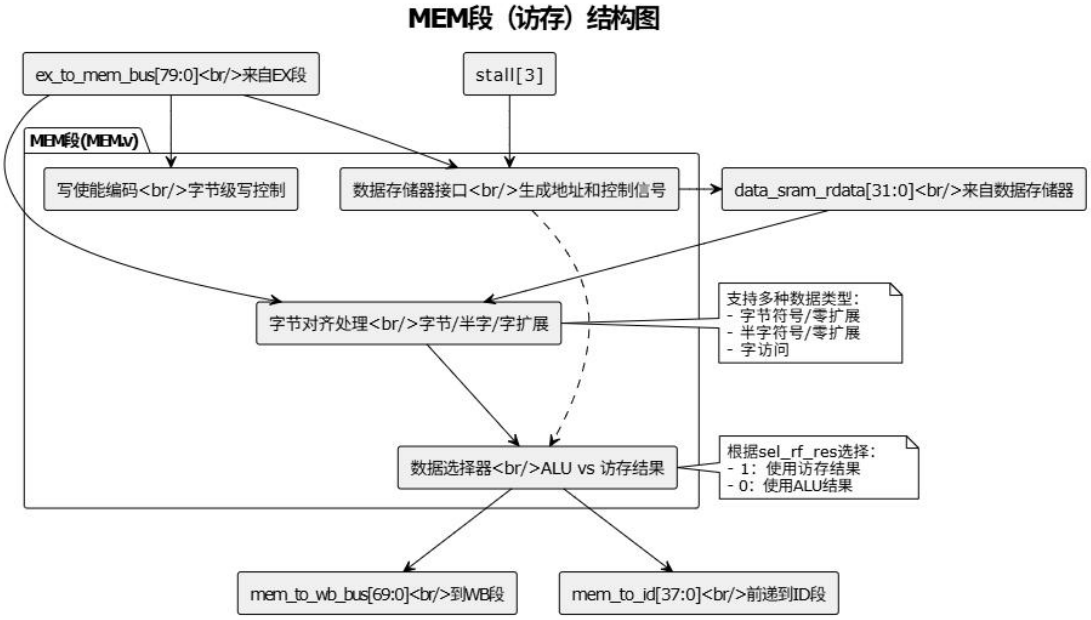
6.4.2 加载数据的译码与扩展（Load Align）

从 data_sram_rdata 读取的 32 位数据不能直接使用，必须根据地址低两位进行对齐：

LBU (Load Byte Unsigned)：截取对应的 8 位并在高位补 0。

LB (Load Byte)：截取 8 位并进行符号位扩展。 这一步骤是通过一个大型的多路选择器配合符号扩展单元实现的。

6.4.3 结构示意图



6.5 WB 段（Write Back，写回）：生命周期的终结

WB 段将流水线运算或访存的最终结果「落盘」到寄存器文件。

6.5.1 功能模块说明

WB 段接收 mem_to_wb_bus，提取出 rf_we（写使能）、rf_waddr（写地址）和 rf_wdata（写数据）。

回传通路：WB 段的输出不仅送往 Register File，还必须回传至 ID 段作为前递路径的最末一环，处理跨度为三条指令的数据相关。

6.5.2 HILO 寄存器管理

针对 mthi，mtlo 指令，本段也负责将结果更新至特有的 HI 和 LO 寄存器，确保乘除法结果的持久化存储。

6.6 流水线暂停与冲突协调机制（Stall Unit）

这是一个贯穿五级流水段的全局模块：

优先级处理：当 ID 段请求暂停（Load-Use）且 EX 段也请求暂停（乘法）时，暂停单元会根据 stall[5:0] 矢量同时冻结多个段。

暂停矢量分配：

stall[0:1]：冻结 IF，ID，用于解决数据冒险。

stall[0:2]：冻结 IF，ID，EX，用于解决多周期执行冒险。

这种分段控制的方法最大限度地减少了不必要的流水线停滞。

七、 实验总结

首先由衷地感谢于老师的细心讲解和各位助教学长们的耐心解答。

从最初处理数据相关时的手忙脚乱，到最后能够熟练运用前递和暂停技术优化处理器性能，我们学到的每一个理论知识在 Verilog 的每一行代码中得到了具象化。这次实验让我们深入理解了计算机组成原理中“流水线”的精髓。尤其

是自主实现乘法器和 AXI 接口的过程，训练了我们解决复杂时序问题的能力。本 CPU 最终以全通过（PASS）的成绩完成了所有既定任务，性能表现稳定，达到了预期设计目标。

李然：

在这次小组实践里,我主要负责数据通路控制模块,如一些 ID 段 forwarding、CTRL 段 stall 逻辑的工作，每一项工作都不轻松。为完成任务，我从各种网站查阅了大量的专业资料和视频，反复进行阅读和理解。在与小组成员的交流协作中，我们能分享一些思路和自己寻找到的能对我们有启发的资料。他们不同的视角和见解，也为我找到很多甚至自己从来没想过的思考方向。我们一同探讨解决方案，共同攻克难关，这样的团队协作的氛围让我很有动力。这次实践让我深刻认识到理论与实践结合的重要性，也体会到团队协作的力量。在未来学习和工作中，我会以此次经历为基石，不断提升自己的专业能力，向前继续奋斗。

王祁萱：

在指令集扩展模块的开发工作中，我真切体会到模块协同的关键价值。这不仅是实现 EX 段 ALU 的控制逻辑,更要衔接 MEM 段的访存指令与基础指令集适配,还得和数据通路模块对齐交互细节,这需要对流水线各阶段协作有精准把控。

理清 EX 段 ALU 控制信号的时序匹配规则至关重要,而 MEM 段访存指令的实现，得兼顾数据读写稳定性与通路兼容性。在 P43 到 P51 的验证中，借助仿真工具定位信号冲突类 Bug，效率提升不少。调试时的每一次信号对齐，都让我对流水线指令流转逻辑理解更深。这段经历不仅强化了模块设计能力，也培养了跨模块协同的全局思维。

王雨菲:

在五级流水线顶层模块的设计工作中，我深刻体会到架构规划的重要性。这不仅是简单地将各个阶段连接起来，更要考虑数据通路、控制逻辑以及如何有效处理各种冒险，这需要对系统有全局性的理解。深入理解 AXI4 协议的五条通道及其时序规范至关重要。在验证过程中,利用仿真工具进行协议检查和错误定位,大大提升了工作效率。性能优化则是一项富有挑战性的任务，它要求我们不仅要关注代码层面的效率，还要从系统层面考虑，识别并消除瓶颈。调试作为整个开发流程中不可或缺的一环，磨练了我的耐心与细致。每解决一个 Bug 都让我对系统的工作原理有了更深刻的理解。这些经历让我受益匪浅,不仅提升了技术能力,也培养了系统性解决问题的思维。

八、参考资料

[1] Kogge P M. The architecture of pipelined computers[M]. New York: McGraw-Hill, 1981.

[2] Thornton J E. Parallel operation in the Control Data 6600[J]. AFIPS Conference Proceedings, 1964, 26: 33-40.
[3] Patterson D A, Hennessy J L. Computer Organization and Design: The Hardware/Software Interface[M]. 5th ed. San Francisco: Morgan Kaufmann, 2013.
[4] Hennessy J L, Patterson D A. Computer Architecture: A Quantitative Approach[M]. 6th ed. San Francisco: Morgan Kaufmann, 2017.
[5] Harris D M, Harris S L. Digital Design and Computer Architecture[M]. 2nd ed. San Francisco: Morgan Kaufmann, 2012.
[6] ARM Ltd. AMBA AXI and ACE Protocol Specification[S]. ARM IHI 0022E, 2013.
[7] Culler D E, Singh J P. Parallel Computer Architecture: A Hardware/Software Approach[M]. San Francisco: Morgan Kaufmann, 1999.
[8] Xilinx Inc. Vivado Design Suite User Guide: Logic Simulation[EB/OL]. UG900, 2023. https://docs.xilinx.com/vivado
[9] 唐朔飞. 计算机组成原理[M]. 第 2 版. 北京: 高等教育出版社, 2008.
[10] 白中英. 计算机组成与体系结构[M]. 第 5 版. 北京: 科学出版社, 2018.
[11] 张晨曦, 王志英. 计算机体系结构[M]. 第 3 版. 北京: 高等教育出版社, 2016.