实验报告成绩:	成绩评定日期:

2023 ~ 2024 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

组长：王彩瑞 20226469

组员：吕佳卉 20226515

报告日期：2025.1.4

一、实验概况

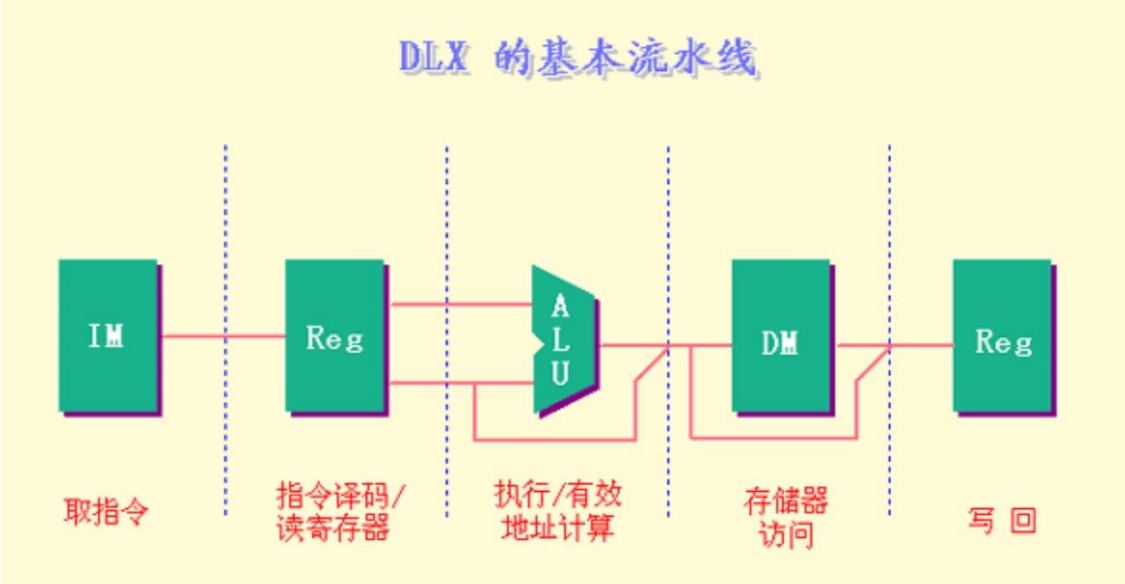
1、工作量

王彩瑞：自制乘法器的设计以及相关信号和寄存器的添加以及 IF、ID、EX 段部分的完善。

吕佳卉：解决数据相关问题。MEM 和 WB 部分完善。

2、总体设计

DLX 流水线的总体设计通过将指令执行划分为 5 个阶段（取指、译码、执行、访存、写回），实现了高效的指令级并行。通过流水线寄存器、数据通路和冲突解决机制，DLX 流水线能够在每个时钟周期完成一条指令的执行（理想情况下），从而显著提高处理器的性能。结构如图所示：



（1）取指令周期（IF）

IR ← Mem[PC]

NPC ← PC＋4

其主要功能是从指令存储器中读取下一条指令，并将其传递给流水线的下一个阶段（通常是译码阶段，ID）。

（2）指令译码/读寄存器周期（ID）

A ← Regs[IR6 ..10]　(Regs[rs])

从寄存器文件中读取源寄存器 rs 的值，并存储在 A 中。

B ← Regs[IR11 ..15]　(Regs[rt])

从寄存器文件中读取源寄存器 rt 的值，并存储在 B 中。

Imm ← (IR16)16 ## IR16 ..31

对指令中的立即数进行符号扩展，并将其存储在 Imm 中，Imm：是一个临时寄存器，用于存储符号扩展后的立即数。

指令译码/读寄存器周期（ID）是 CPU 流水线的第二个阶段，其主要功能是对取指令周期（IF）取到的指令进行译码，并从寄存器文件中读取操作数。

（3）执行/有效地址计算周期（EX）

• ALUOutput ← A＋Imm 存储器访问（load 和 store）

• ALUOutput ← A op B 寄存器－寄存器 ALU 操作

• ALUOutput ← A op Imm 寄存器－立即值 ALU 操作

• ALUOutput ← NPC＋Imm 分支操作

Cond ← (A op 0)

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。执行/有效地址计算周期涉及以下关键组件：算术逻辑单元（ALU）：用于执行算术和逻辑运算。加法器：用于计算有效地址和分支目标地址。多路选择器（Multiplexer）：用于选择不同的操作数或结果。

（4）存储器访问 / 分支完成周期（MEM）

存储器访问（load 和 store）

LMD ← Mem[ALUOutput]或 Mem [ALUOutput] ← B

分支操作

if（cond）PC ← ALUOutput

else PC ← NPC

MEM 阶段的核心任务是：执行内存的读写操作（Load/Store）。处理分支指令，决定下一条指令的地址（PC 更新）。

（5）写回周期（WB）

寄存器－寄存器型 ALU 指令

Regs[IR16 ..20] (rd)← ALUOutput

寄存器－立即值型 ALU 指令

Regs[IR11 ..15] (rt)← ALUOutput

Load 指令

Regs[IR11 ..15] (rt)← LMD

写回阶段（WB）的核心任务是将 ALUOutput 或从内存中读取的数据写回到目标寄存器中，具体目标寄存器由指令中的字段（rd 或 rt）决定。WB 阶段是流水线的最后一步，确保指令的执行结果被正确保存，供后续指令使用。

其中，分支指令需要 4 个时钟周期（移到 ID 段，只需 2 个周期），store 指令需要 4 个时钟周期，其它指令需要 5 个时钟周期。

3、MIPS 指令格式

MIPS 指令系统采用固定长度的 32 位指令格式，主要包括三种类型：立即数型（I-Type）、寄存器型（R-Type）和跳转型（J-Type）。每种指令类型具有独特的字段结构，以适应不同的操作需求。

3.1 立即数型指令（I-Type）

I-Type 指令主要用于加载/存储操作、立即数运算以及条件分支。其结构如下：

操作码（6 位）：定义指令类型。

rs（5 位）：源寄存器，通常用于存储基地址或操作数。

rt（5 位）：目标寄存器，用于存储结果或数据。

立即数（16 位）：提供常数或偏移量。

典型指令示例：

加载指令（lw）：访存有效地址：Regs[rs] + immediate。从存储器读取的数据存入寄存器 rt。

存储指令（sw）：访存有效地址：Regs[rs] + immediate。将寄存器 rt 中的数据存入存储器。

分支指令（beq）：转移目标地址：PC + 4 + (immediate << 2)。若 Regs[rs] == Regs[rt]，则跳转到目标地址。

3.2 寄存器型指令（R-Type）

R-Type 指令用于寄存器间的算术逻辑运算，其格式如下：

操作码（6 位）：固定为 000000，表示 R-Type 指令。

rs、rt（各 5 位）：源寄存器。

rd（5 位）：目标寄存器。

shamt（5 位）：移位量，用于移位指令。

func（6 位）：功能码，指定具体操作。

典型指令示例：

加法指令（add）：功能码：100000；操作：Regs[rd] ← Regs[rs] + Regs[rt]

逻辑与指令（and）：功能码：100100；操作：Regs[rd] ← Regs[rs] & Regs[rt]

移位指令（sll）：功能码：000000；操作：Regs[rd] ← Regs[rt] << shamt

3.3 跳转型指令（J-Type）

J-Type 指令用于无条件跳转和子程序调用，其结构如下：

操作码（6 位）：定义跳转类型。目标地址（26 位）：跳转目标的偏移量，与 PC 值结合形成最终地址。

典型指令示例：

跳转指令（j）：操作码：000010；跳转目标地址：(PC & 0xF0000000) | (target << 2)

跳转并链接指令（jal）：操作码：000011；跳转目标地址同上，同时将返回地址（PC+4）存入$ra 寄存器。

4 流水线设计概述

流水线设计是 MIPS 处理器的核心，通过将指令执行过程划分为多个阶段（如取指、译码、执行、访存、写回），提高指令执行的并行性。

4.1 流水线阶段

取指（IF）：从指令存储器中读取指令。

译码（ID）：解码指令并读取寄存器操作数。

执行（EX）：执行算术逻辑运算或计算地址。

访存（MEM）：访问数据存储器（加载/存储指令）。

写回（WB）：将结果写回寄存器。

4.2 流水线冲突处理

数据冲突：通过数据转发（forwarding）和流水线暂停（stall）解决。

5 指令完成情况

本实验实现了 MIPS-32 指令集中的 49 条指令，涵盖算术运算、逻辑运算、移位操作、分支跳转、数据移动和访存操作。具体指令如下：

5.1 已完成指令

算术运算指令：add, addi, addu, sub, subu, mult, multu, div, divu

逻辑运算指令：and, andi, or, ori, xor, xori, nor

移位指令：sll，srl，sra，sllv，srlv，srav

分支跳转指令：beq，bne，j，jal，jr，jalr，bgez，bltz，bgtz，blez，bgezal，bltzal

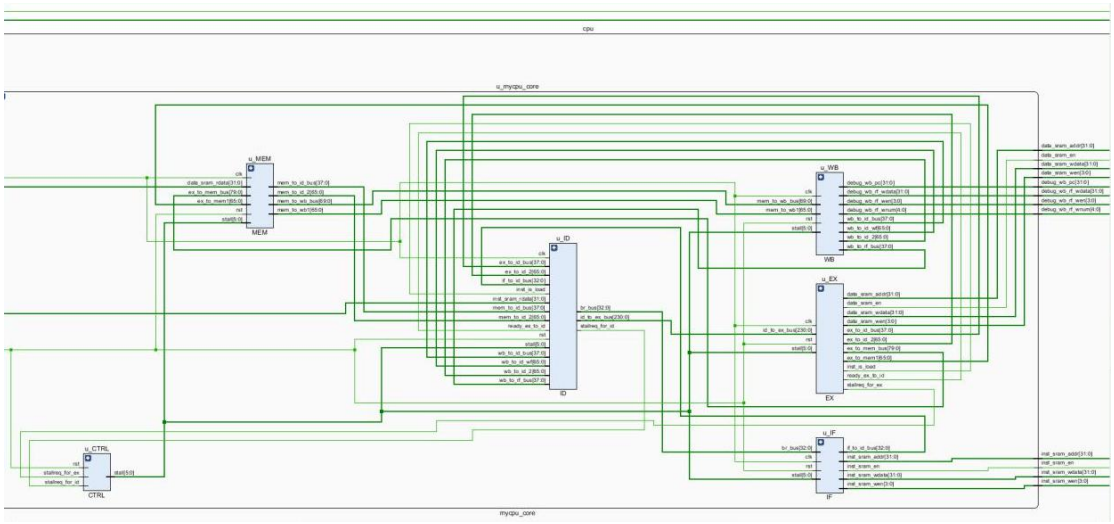
数据移动指令：mfhi，mflo，mthi，mtlo

访存指令：lw，sw，lb，lbu，lh，lhu，sb，sh

5.2 实验成果

成功通过第 64 个测试点。

实现了一个 32 时钟周期的乘法器，支持有符号和无符号乘法运算。



6、程序运行环境及使用工具

操作系统：Windows 10 64 位。

开发平台：Vivado 2019.2。

编程语言：Verilog HDL 硬件描述语言。

二、单个流水段说明

1、IF 段

1.1. 主要任务

IF 段是 CPU 流水线的第一个阶段，主要任务是从指令存储器中读取指令，并更新程序计数器（PC）。它的核心工作可以总结为以下几点：根据 PC 的当前值，从指令存储器中读取指令。将读取的指令传递给下一个阶段（ID 段）进行译码和执行。更新 PC 的值，指向下一条指令或跳转目标地址。

1.2 工作流程

IF 段的工作流程可以分为以下几个步骤：

IF 段以 PC 的当前值为地址，从指令存储器中读取对应的指令。如果没有分支跳转，PC 的值会加 4（假设每条指令占 4 个字节），指向下一条指令。如果有分支跳转（如条件跳转或无条件跳转），PC 的值会被更新为跳转目标地址。IF 段会根据流水线暂停信号（stall）决定是否继续工作。如果流水线暂停，PC 的值会保持不变，指令读取也会暂停。IF 段将当前的 PC 值和指令存储器使能信号通过总线（if_to_id_bus）传给 ID 段，同时将 PC 值作为地址传给指令存储器，确保指令存储器能返回正确的指令。

1.3. 端口介绍

输入端口：

clk：时钟信号，控制模块的运行节奏。rst：复位信号，高电平有效。复位时，PC 会被设为一个默认值（如 32'hbfbf_fffc）。stall[5:0]：流水线暂停信号。如果 stall[0]是 NoStop，IF 段正常工作；否则，PC 和指令读取会暂停。br_bus[32:0]：分支跳转信号，包含分支使能信号（br_e）和分支目标地址（br_addr）。如果 br_e 为高电平，PC 会跳转到 br_addr。

输出端口：

if_to_id_bus[32:0]：传给 ID 段的总线信号，包含指令存储器使能信号（ce_reg）和当前 PC 值（pc_reg）。inst_sram_addr[31:0]：指令存储器地址，就是当前的 PC 值。inst_sram_en：指令存储器使能信号，高电平有效。当它为高电平时，指令存储器会根据地址返回对应的指令。inst_sram_wdata[31:0]：指令存储器写数据，始终为 0（因为 IF 段只读不写）。inst_sram_wen[3:0]：指令存储器写使能信号，始终为 0（因为 IF 段只读不写）。

1.4. 具体工作细节

复位阶段：当 rst 信号为高电平时，PC 会被设为一个默认值（如 32'hbfbf_fffc），同时指令存储器使能信号（ce_reg）会被关闭。正常取指令：每个时钟周期，IF 段会检查 stall[0] 信号。如果 stall[0] 是 NoStop，PC 的值会更新为 next_pc。next_pc 的值由分支跳转信号决定：如果有跳转（br_e 为高电平），next_pc 就是 br_addr；否则，next_pc 就是 pc_reg + 4。读取指令：IF 段把 pc_reg 作为地址传给指令存储器，并生成 inst_sram_en 信号，让指令存储器返回对应的指令。返回的指令和当前 PC 值会通过 if_to_id_bus 传给 ID 段。流水线暂停：如果 stall[0] 不是 NoStop，PC 的值会保持不变，指令读取也会暂停。

1.5. 总结

IF 段是 CPU 流水线的起点，负责从指令存储器中读取指令并更新 PC。它通过处理分支跳转和流水线暂停信号，确保指令的正确读取和传递。IF 段的输出为后续阶段（ID 段）提供了必要的指令和地址信息，是流水线正常运行的关键。

2、ID 段

2.1 整体功能说明

指令译码/读寄存器周期（ID 段）是 CPU 流水线中的一个关键阶段，主要负责对从指令存储器中获取的指令进行译码，并根据指令中的寄存器地址从通用寄存器组中读取相应的寄存器值。如果指令中包含立即数，ID 段还需要对立即数进行符号扩展或无符号扩展。此外，如果指令是跳转指令且满足跳转条件，ID 段会计算跳转地址，并通过 br_bus 总线将跳转地址传回 IF 段，以便 IF 段能够快速获取下一条指令的地址。同时，ID 段还会将各种控制信号和操作数通过 id_to_ex_bus 总线传递给下一阶段的 EX 段。ID 段的一个显著特点是指令译码和读寄存器操作是并行进行的。这种并行性得益于 DLX 指令格式中操作码的固定位置，这种技术也称为固定字段译码。

2.2 端口介绍

输入端口：clk：时钟信号。rst：复位信号。

stall[5:0]:流水线暂停信号。ex_to_id_bus[37:0]:从 EX 段传递到 ID 段的总线信号。mem_to_id_bus[37:0]：从 MEM 段传递到 ID 段的总线信号。wb_to_id_bus[37:0]：从 WB 段传递到 ID 段的总线信号。ex_to_id_2[65:0]：从 EX 段传递到 ID 段的第二个总线信号。mem_to_id_2[65:0]：从 MEM 段传递到 ID 段的第二个总线信号。wb_to_id_2[65:0]：从 WB 段传递到 ID 段的第二个总线信号。if_to_id_bus[32:0]：从 IF 段传递的当前指令的 PC 值。inst_sram_rdata[31:0]：从指令存储器读取的指令数据。inst_is_load：指示当前指令是否为加载指令。wb_to_rf_bus[37:0]：从 WB 段传递到寄存器文件的总线信号。wb_to_id_wf[65:0]：从 WB 段传递到 ID 段的写回信号。ready_ex_to_id：EX 段到 ID 段的就绪信号。

输出端口：

br_bus[32:0]：跳转信号总线，包含跳转地址和跳转使能信号。id_to_ex_bus[230:0]:ID 段传递到 EX 段的总线信号,包含控制信号和操作数。stallreq_for_id：ID 段的暂停请求信号。

2.3 信号介绍

rst：复位信号，用于将 ID 段的状态重置为初始值。clk：时钟信号，用于同步 ID 段的操作。if_to_id_bus[32:0]：从 IF 段传递的当前指令的 PC 值，其中 if_to_id_bus[31:0] 是 PC 值。inst_sram_rdata[31:0]：从指令存储器读取的指令数据。如果 ID 段未被暂停，该值会被赋给 inst 变量，并进行指令译码。

ex_to_id_bus[37:0]、mem_to_id_bus[37:0]、wb_to_id_bus[37:0]：这些总线信号用于解决数据相关问题。如果当前指令需要读取尚未写入寄存器的值，EX 段、MEM 段和 WB 段会提前将这些值发送给 ID 段，ID 段再将这些值传递给寄存器文件模块 regfile.v，以确保读取到正确的寄存器值。

stall[5:0]：流水线暂停信号。如果 stall[2] 为高电平，ID 段会暂停当前操作，并将 if_to_id_bus_r 的值清零，以确保下一个时钟周期不会接收到错误的值。ready_ex_to_id：EX 段传递的信号，用于指示乘除法操作是否完成。如果乘除法操作未完成，ID 段会暂停流水线，并保存当前指令的值，直到操作完成。wb_to_id_wf[65:0] 和 wb_to_rf_bus[37:0]：从 WB 段传递的写回信号，用于将数据写回寄存器文件。inst_is_load：EX 段传递的信号，用于指示当前指令是否为加载指令（LW）。如果是加载指令，且加载的目标寄存器与 ID 段当前指令的源寄存器相同，ID 段会发出暂停请求信号 stallreq_for_id，以避免数据冲突。br_bus[32:0]：跳转信号总线，包含跳转地址和跳转使能信号。ID 段会根据跳转指令的计算结果，将跳转地址传递给 IF 段。id_to_ex_bus[230:0]：ID 段传递到 EX 段的总线信号，包含以下信息：

当前指令的 PC 值和指令数据。ALU 操作的控制信号 alu_op。ALU 操作数的选择信号 sel_alu_src1 和 sel_alu_src2。存储器访问使能信号 data_ram_en 和写使能信号 data_ram_wen[3:0]。寄存器写使能信号 rf_we 和写地址。寄存器读数据 rdata1 和 rdata2。高低位寄存器的读写信号 lo_hi_r[1:0] 和 lo_hi_w[1:0]。高低位寄存器的值 lo_o 和 hi_o。存储器读信号 data_ram_read[3:0]。

2.4 regfile 功能模块介绍

regfile.v 是 ID 段中的一个重要模块，负责管理通用寄存器。reg_array[31:0]和用于存储乘除法结果的高低位寄存器 hi 和 lo。

输入信号：

r_hi_we 和 r_lo_we：高低位寄存器的读使能信号。raddr1[4:0] 和 raddr2[4:0]：要读取的寄存器地址。w_hi_we 和 w_lo_we：高低位寄存器的写使能信号。hi_i 和 lo_i：要写入高低位寄存器的数据。we：寄存器写使能信号。waddr[4:0]：要写入的寄存器地址。wdata[31:0]：要写入寄存器的数据。

输出信号：

hi_o[31:0] 和 lo_o[31:0]：高低位寄存器的输出数据。rdata1[31:0] 和 rdata2[31:0]：从寄存器组读取的数据。

功能说明：

regfile.v 模块通过 raddr1 和 raddr2 从寄存器组中读取数据，并通过 rdata1 和 rdata2 返回给 ID 段。如果当前指令需要读取的寄存器地址与 EX、MEM 或 WB 段即将写入的寄存器地址相同，regfile.v 会提前将写入的值赋给

rdata1 或 rdata2，从而解决数据相关问题。高低位寄存器 hi 和 lo 用于存储乘除法指令的结果，regfile.v 会根据 w_hi_we 和 w_lo_we 信号将数据写入高低位寄存器，或根据 r_hi_we 和 r_lo_we 信号读取高低位寄存器的值。

2.5 数据冲突解决

在 regfile.v 中，数据冲突的解决是通过提前将 EX、MEM 和 WB 段的数据传递给 ID 段来实现的。如果当前指令需要读取的寄存器地址与即将写入的寄存器地址相同，regfile.v 会直接将写入的值赋给 rdata1 或 rdata2，从而避免读取到错误的值。这种机制有效地解决了数据相关问题，确保了流水线的正确执行。

2.6 总结

ID 段是 CPU 流水线中的关键阶段，负责指令译码、寄存器读取、立即数扩展以及跳转地址计算等功能。通过并行处理指令译码和寄存器读取，ID 段能够高效地完成其任务。同时，通过与 EX、MEM 和 WB 段的协作，ID 段还能够解决数据冲突问题，确保流水线的正确执行。regfile.v 模块作为 ID 段的核心组件，负责管理寄存器组和乘除法结果寄存器，并通过提前传递数据的方式解决了数据相关问题。

3、EX 段

3.1 主要任务

执行阶段（EX）的核心功能有以下几个方面：完成算术逻辑运算（ALU 操作）、处理乘法和除法指令、控制数据存储器的访问（读/写操作）、处理流水线停顿请求，避免数据冒险、将执行结果传递到下一阶段（MEM 阶段）和回传到译码阶段（ID 阶段）。

3.2 工作流程

从译码阶段（ID）接收 id_to_ex_bus 总线信号，解析出操作数、控制信号和指令信息。再根据指令类型（如乘法、除法、加载/存储指令），执行相应的操作。ALU 运算：根据 alu_op 和操作数（alu_src1 和 alu_src2）执行算术逻辑运算。乘法和除法运算：检测乘法指令（mult 和 multu）和除法指令（div 和 divu）。调用乘法模块（mul_plus）和除法模块（div）进行计算。将结果写入 HI 和 LO 寄存器。再根据指令生成存储器地址（data_sram_addr）和写入数据（data_sram_wdata）。对于流水线控制：检测加载指令（inst_is_load），避免数据冒险。最后根据乘法和除法的完成状态（mul_ready_i 和 div_ready_i），请求流水线停顿（stallreq_for_ex）。将执行结果打包到 ex_to_mem_bus 总线，传递到访存阶段（MEM）。将部分结果打包到 ex_to_id_bus 和 ex_to_id_2 总线，回传到译码阶段（ID）。

3.3 端口介绍

输入端口：

clk：时钟信号、rst：复位信号、stall：流水线停顿信号，用于控制流水线的运行、id_to_ex_bus：从译码阶段（ID）传递到执行阶段（EX）的总线信号，包含操作数、控制信号和指令信息。

输出端口：

ex_to_mem_bus：从执行阶段（EX）传递到访存阶段（MEM）的总线信号，包含执行结果和控制信号、ex_to_id_bus：从执行阶段（EX）回传到译码阶段（ID）的总线信号,用于数据旁路、data_sram_en:数据存储器使能信号、data_sram_wen:数据存储器写使能信号、data_sram_addr：数据存储器地址、data_sram_wdata：数据存储器写入数据、inst_is_load：加载指令标志，用于流水线控制、stallreq_for_ex：执行阶段的停顿请求信号、ex_to_meml：传递 HI 和 LO 寄存器的写使能信号和写数据到访存阶段（MEM）、ex_to_id_2：传递 HI 和 LO 寄存器的写使能信号和写数据到译码阶段（ID）、ready_ex_to_id：指示乘法和除法操作是否完成。

3.4 信号介绍

控制信号：

alu_op：ALU 操作控制信号。

sel_alu_src1 和 sel_alu_src2：ALU 操作数选择信号。

data_ram_en 和 data_ram_wen：数据存储器使能和写使能信号。

rf_we：寄存器文件写使能信号。

rf_waddr：寄存器文件写地址。

sel_rf_res：选择寄存器文件写入结果的信号。

数据信号：

rf_rdata1 和 rf_rdata2：寄存器文件读取的数据。

alu_result：ALU 计算结果。

ex_result：执行阶段的最终结果。

mul_result 和 div_result：乘法和除法的结果。

hi_i 和 lo_i：HI 和 LO 寄存器的写数据。

hi_o 和 lo_o：HI 和 LO 寄存器的读数据。

流水线控制信号：

stallreq_for_div：除法操作的停顿请求信号。

ready_ex_to_id：指示乘法和除法操作是否完成。

3.5 mul_plus 功能模块介绍

这个模块实现了一个支持有符号乘法和无符号乘法的乘法器，通过移位和累加的方式完成乘法运算。通过流水线进行移位乘法的计算，减少了时钟周期。它的计算原理如下：

可以先举一个简单的例子说明：

```
G = G << 1;      // 即 G * 2
B = (B << 2) + B; // 即 B * 5
R = (R << 6) + (R << 3) + (R << 2) + R  // 即 R * 77
```

再推广到我们的流水线中：寄存器接收两个 32 位操作数（opdata1_i 和 opdata2_i），并根据 mul_sign 信号判断是否进行有符号乘法。模块首先计算操作数的符号位和绝对值，然后通过被乘数左移、乘数右移的方式逐位累加部分积，最终生成 64 位的乘法结果（result_o）。如果是有符号乘法且结果为负，模块会对结果取补码。此外，模块还通过 ready_o 信号指示乘法操作是否完成，当乘数为 0 时，ready_o 置为高电平，表示乘法完成。该模块通过硬件逻辑实现了高效的乘法运算，适用于处理器中的算术逻辑单元（ALU）。通过 EX 段中的 mul 模块定义与自定义乘法的 mul_plus 相连接，具体代码如下：

```
mul_plus u_mul_plus(
    .clk        (clk            ),
    .start_i    (mul_begin),
    .mul_sign   (mul_signed),
    .opdata1_i  (  rf_rdata1    ),
    .opdata2_i  (  rf_rdata2    ),
    .result_o   (mul_result     ),
    .ready_o    (mul_ready_i      )
);
```

3.6 总结

该模块是处理器的执行阶段（EX），是处理器流水线的核心部分，负责完成算术逻辑运算、乘法和除法操作，并控制数据存储器的访问。通过流水线控制和数据旁路机制，解决了数据冒险问题，提高了处理器的效率。

4、MEM 段

4.1 主要任务

访存阶段（MEM）的核心功能主要有以下几个方面：处理数据存储器的访问（读/写操作）。将执行阶段（EX）的结果传递到写回阶段（WB）。将 HI 和 LO 寄存器的写使能信号和写数据传递到写回阶段（WB）和译码阶段（ID）。通过数据旁路机制，解决流水线中的数据冒险问题。

4.2 工作流程

输入信号处理：从执行阶段（EX）接收 ex_to_mem_bus 总线信号，解析出存储器访问控制信号、寄存器写使能信号和写数据。从执行阶段（EX）接收 ex_to_mem1 总线信号，解析出 HI 和 LO 寄存器的写使能信号和写数据。

数据存储器访问：根据 data_ram_en 和 data_ram_wen 信号，控制数据存储器的读/写操作。根据 data_ram_read 信号，处理不同字节宽度的数据读取。

结果传递：将存储器读取的数据或执行阶段的结果打包到 mem_to_wb_bus 总线，传递到写回阶段（WB）。将 HI 和 LO 寄存器的写使能信号和写数据打包到 mem_to_wb1 总线，传递到写回阶段（WB）。将 HI 和 LO 寄存器的写使能信号和写数据打包到 mem_to_id_2 总线，传递到译码阶段（ID）。将寄存器文件的写使能信号、写地址和写数据打包到 mem_to_id_bus 总线，传递到译码阶段（ID）。

4.3 端口介绍

输入端口：clk：时钟信号、rst：复位信号、stall：流水线停顿信号，用于控制流水线的运行、ex_to_mem_bus：从执行阶段（EX）传递到访存阶段（MEM）的总线信号，包含存储器访问控制信号、寄存器写使能信号和写数据 data_sram_rdata：从数据存储器读取的数据、ex_to_mem1：从执行阶段（EX）传递到访存阶段（MEM）的总线信号，包含 HI 和 LO 寄存器的写使能信号和写数据。

输出端口：mem_to_wb_bus：从访存阶段（MEM）传递到写回阶段（WB）的总线信号，包含寄存器写使能信号、写地址和写数据。mem_to_id_bus：从访存阶段（MEM）回传到译码阶段（ID）的总线信号，包含寄存器写使能信号、写地址和写数据。mem_to_wb1：从访存阶段（MEM）传递到写回阶段（WB）的总线信号，包含 HI 和 LO 寄存器的写使能信号和写数据。mem_to_id_2：从访存阶段（MEM）回传到译码阶段（ID）的总线信号，包含 HI 和 LO 寄存器的写使能信号和写数据。

4.4 信号介绍

控制信号：

data_ram_en：数据存储器使能信号。

data_ram_wen：数据存储器写使能信号。

data_ram_read：数据存储器读取模式信号。

rf_we：寄存器文件写使能信号。

rf_waddr：寄存器文件写地址。

sel_rf_res：选择寄存器文件写入结果的信号。

数据信号：

ex_result：执行阶段的结果。

mem_result：存储器读取的数据。

rf_wdata：寄存器文件写入的数据。

hi_i 和 lo_i：HI 和 LO 寄存器的写数据。

w_hi_we 和 w_lo_we：HI 和 LO 寄存器的写使能信号。

流水线控制信号：

stall：流水线停顿信号，用于控制流水线的运行。

4.5 总结

该模块是处理器的访存阶段（MEM），负责处理数据存储器的访问，并将结果传递到写回阶段(WB)和译码阶段(ID)。通过 mem_to_wb_bus 和 mem_to_id_bus 总线，实现了结果的多阶段传递。通过 mem_to_wb1 和 mem_to_id_2 总线，实现了 HI 和 LO 寄存器写操作的流水线传递。

5、WB 段

5.1 主要任务

将访存阶段（MEM）的结果写回寄存器文件（Register File）。提供调试信息，用于跟踪写回阶段的寄存器写入操作。

5.2 工作流程

输入信号处理：从访存阶段（MEM）接收 mem_to_wb_bus 总线信号，解析出程序计数器（PC）、寄存器写使能信号、写地址和写数据。

写回操作：根据 rf_we 信号，判断是否需要将数据写入寄存器文件。将写地址和写数据传递到寄存器文件。

调试信息输出：将写回阶段的 PC、寄存器写使能信号、写地址和写数据输出到调试端口，用于调试和跟踪。

5.3 端口介绍

输入端口：clk：时钟信号 r、st：复位信号、stall：流水线停顿信号，用于控制流水线的运行、mem_to_wb_bus：从访存阶段（MEM）传递到写回阶段（WB）的总线信号，包含 PC、寄存器写使能信号、写地址和写数据。

输出端口：wb_to_rf_bus：从写回阶段（WB）传递到寄存器文件的总线信号，包含寄存器写使能信号、写地址和写数据。debug_wb_pc：写回阶段的 PC，用于调试。debug_wb_rf_wen：寄存器写使能信号，用于调试。debug_wb_rf_wnum：寄存器写地址，用于调试。debug_wb_rf_wdata：寄存器写数据，用于调试。

5.4 信号介绍

控制信号：

rf_we：寄存器文件写使能信号，表示是否需要将数据写入寄存器文件。

数据信号：

wb_pc：写回阶段的程序计数器（PC）。

rf_waddr：寄存器文件写地址。

rf_wdata：寄存器文件写数据。

调试信号：

debug_wb_pc：写回阶段的 PC，用于调试。

debug_wb_rf_wen：寄存器写使能信号，用于调试。

debug_wb_rf_wnum：寄存器写地址，用于调试。

debug_wb_rf_wdata：寄存器写数据，用于调试。

5.5 总结

该模块是处理器的写回阶段（WB），负责将访存阶段（MEM）的结果写回寄存器文件。通过 wb_to_rf_bus 总线，将寄存器写使能信号、写地址和写数据传递到寄存器文件。同时还可以提供调试信息，用于跟踪写回阶段的寄存器写入操作。是处理器流水线的最后阶段。

三、实验感受及改进意见

1、王彩瑞

在设计 CPU 的过程中，我深刻体会到理论知识与实际操作的结合非常重要。书本上的知识只是一个基础，真正动手设计时，会遇到许多细节问题，需要灵活运用理论知识来解决。将 CPU 划分为多个模块（如取指、译码、执行、访存、写回）可以简化设计过程。每个模块只需要关注自己的功能，模块之间的接口设计清晰后，整个系统的复杂性会大大降低。调试是 CPU 设计中最耗时的部分，但也是最关键的部分。通过仿真工具观察信号波形，可以快速定位问题所在。编

写全面的测试程序，覆盖各种边界情况，是确保 CPU 正确性的重要手段。CPU 设计是一个需要耐心和细致的工作。每一个信号、每一条指令都需要仔细设计和测试，稍有不慎就可能导致整个系统无法正常工作。

通过动手实践我对 CPU 有了更深刻的理解，在本次实验中一门新的编程语言以及编译环境都为实验的完成带来了很大的考验，好在通过询问主教与同学，遇到的问题都得到了解决。

2、吕佳卉

本次实验围绕 DLX 流水线的设计与实现展开，工作量较大，涉及流水线的五个阶段（取指、译码、执行、访存、写回）以及 MIPS 指令格式的理解与实现。在实验中，流水线的总体设计让我认识到指令级并行对提升处理器性能的关键作用。通过将指令执行划分为多个阶段，流水线能够在每个时钟周期完成一条指令的执行（理想情况下），从而显著提高效率。然而，流水线中的冲突问题也让我意识到，设计高效的冲突解决机制是确保流水线稳定运行的关键。在实现过程中，流水线的设计让我对指令的读取、译码以及寄存器操作有了更深入的理解。

总体而言，本次实验不仅让我掌握了流水线的基本原理和实现方法，还让我认识到硬件设计中的复杂性与挑战。通过不断调试与优化，我逐渐理解了如何在实际应用中平衡性能与稳定性，这对我的硬件设计能力提升具有重要意义。

四、参考资料

[1] 雷思磊.《自己动手做 cpu_雷思磊》[M/CD].