实验报告成绩:	成绩评定日期:

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

1.分工

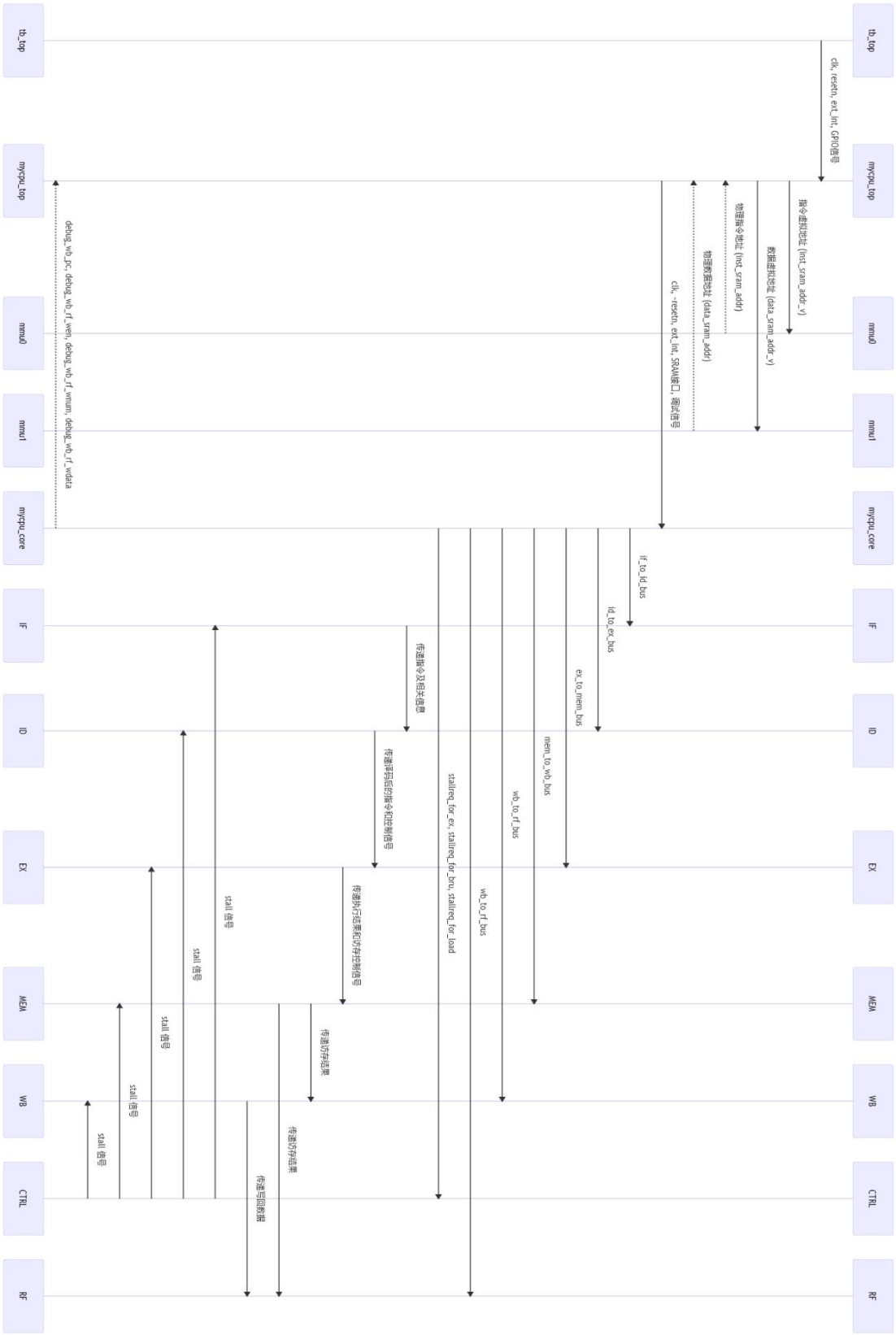
姓名	分工	工作量占比
刘庆然	整体设计以及报告编写	65%
马智鸿	参与部分实现、参与报告编写	35%

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


tb_top
mycpu_top
mmu0
mmu1
mycpu_core
IF
ID
EX
MEM
WB
CTRL
RF
clk, resetn, ext_int, GPIO信号
指令虚拟地址 (inst_sram_addr_v)
数据虚拟地址 (data_sram_addr_v)
物理指令地址 (inst_sram_addr)
物理数据地址 (data_sram_addr)
clk, resetn, ext_int, SRAM接口, 调试信号
debug_wb_pc, debug_wb_rf_wen, debug_wb_rf_wnum, debug_wb_rf_wdata
if_to_id_bus
id_to_ex_bus
ex_to_mem_bus
mem_to_wb_bus
wb_to_rf_bus
wb_to_rf_bus
stallreq_for_ex, stallreq_for_bru, stallreq_for_load
传递指令及相关信息
传递译码后的指令和控制信号
传递执行结果和访存控制信号
传递访存结果
传递访存结果
传递写回数据
stall 信号
stall 信号
stall 信号
stall 信号

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。

二、 流水线相应阶段说明

1.IF 模块

IF 模块输入输出：

序号	接口名	宽度	输入/输出	作用
1	clk	1	输入	时钟信号，驱动模块内部的时序逻辑
2	rst	1	输入	复位信号，高电平有效，用于初始化模块
3	stall	`StallBus`	输入	停顿信号总线，控制流水线的暂停与恢复
4	br_bus	`BR_WD`	输入	分支信息总线，包含分支有效标志和分支地址
5	if_to_id_bus	`IF_TO_ID_WD`	输出	传递给 ID 阶段的数据总线，包含使能信号和 PC
6	inst_sram_en	1	输出	指令存储器使能信号，控制是否进行读操作
7	inst_sram_wen	4	输出	指令存储器写使能信号，固定为0，不进行写操作
8	inst_sram_addr	32	输出	指令存储器地址，根据当前 PC 值生成
9	inst_sram_wdata	32	输出	指令存储器写数据，固定为0，不进行写操作

IF 模块详细功能说明

序号	信号名	宽度	输入/输出	作用
1	pc_reg	32	内部	内部寄存器，保存当前的程序计数器（PC）值
2	ce_reg	1	内部	内部寄存器，保存指令存储器使能信号
3	next_pc	32	内部	计算出的下一个程序计数器值
4	br_e	1	内部	分支是否有效的标志
5	br_addr	32	内部	分支目标地址
6	if_to_id_bus	`IF_TO_ID_WD`	输出	传递给译码阶段的数据总线，包含使能信号和当前 PC 值
7	inst_sram_en	1	输出	指令存储器使能信号，控制是否进行读操作
8	inst_sram_wen	4	输出	指令存储器写使能信号（固定为0，表示不进行写操作）
9	inst_sram_addr	32	输出	指令存储器地址，根据当前 PC 值生成
10	inst_sram_wdata	32	输出	指令存储器写数据（固定为0，表示不进行写操作）
11	debug_wb_pc	32	输出	调试用：写回阶段的 PC
12	debug_wb_rf_wen	4	输出	调试用：寄存器写使能信号
13	debug_wb_rf_wnum	5	输出	调试用：寄存器写地址
14	debug_wb_rf_wdata	32	输出	调试用：寄存器写数据

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

序号	接口名	宽度	输入/输出	作用
1	clk	1	输入	时钟信号，驱动模块内部的时序逻辑
2	rst	1	输入	复位信号，高电平有效，用于初始化模块
3	stall	`StallBus`	输入	停顿信号总线，控制流水线的暂停与恢复
4	if_to_id_bus	`IF_TO_ID_WD`	输入	从 IF 阶段到 ID 阶段的数据总线
5	inst_sram_rdata	32	输入	从指令存储器读取的指令数据
6	ex_id	1	输入	EX阶段标识信号，用于数据转发相关操作
7	wb_to_rf_bus	`WB_TO_RF_WD`	输入	从 WB 阶段到寄存器文件的数据总线
8	ex_to_rf_bus	`EX_TO_RF_WD`	输入	从 EX 阶段到寄存器文件的数据总线
9	mem_to_rf_bus	`MEM_TO_RF_WD`	输入	从 MEM 阶段到寄存器文件的数据总线
10	ex_hi_lo_bus	66	输入	从 EX 阶段传递的 HI/LO 寄存器相关信号
11	id_hi_lo_bus	72	输出	ID 阶段传递给 EX 阶段的 HI/LO 寄存器相关信号
12	id_load_bus	`LoadBus`	输出	ID 阶段传递的 Load 信号
13	id_save_bus	`SaveBus`	输出	ID 阶段传递的 Save 信号
14	stallreq	1	输出	停顿请求信号，通知流水线暂停
15	stallreq_for_bru	1	输出	分支指令单元发出的停顿请求信号
16	id_to_ex_bus	`ID_TO_EX_WD`	输出	从 ID 阶段到 EX 阶段的数据总线
17	br_bus	`BR_WD`	输出	分支跳转相关的信号总线

ID 模块详细功能说明:

序号	信号名	宽度	输入/输出	作用
1	if_to_id_bus_r	`IF_TO_ID_WD`	内部	暂存从 IF 阶段传递过来的数据总线
2	inst	32	内部	当前指令，根据使能信号和标志位选择指令
3	id_pc	32	内部	当前指令的程序计数器（PC）值
4	ce	1	内部	使能信号，从 `if_to_id_bus_r` 解码得到
5	flag	1	内部	标志位，用于处理停顿请求
6	buf_inst	32	内部	缓存指令，用于处理停顿期间的指令
7	wb_rf_we	1	内部	来自 WB 阶段的寄存器文件写使能信号
8	wb_rf_waddr	5	内部	来自 WB 阶段的寄存器文件写地址
9	wb_rf_wdata	32	内部	来自 WB 阶段的寄存器文件写数据
10	ex_rf_we	1	内部	来自 EX 阶段的寄存器文件写使能信号
11	ex_rf_waddr	5	内部	来自 EX 阶段的寄存器文件写地址
12	ex_rf_wdata	32	内部	来自 EX 阶段的寄存器文件写数据
13	mem_rf_we	1	内部	来自 MEM 阶段的寄存器文件写使能信号
14	mem_rf_waddr	5	内部	来自 MEM 阶段的寄存器文件写地址
15	mem_rf_wdata	32	内部	来自 MEM 阶段的寄存器文件写数据
16	rs_eq_rt	1	内部	rs 寄存器值是否等于 rt 寄存器值
17	rs_ge_z	1	内部	rs 寄存器值是否大于等于 0
18	rs_gt_z	1	内部	rs 寄存器值是否大于 0
19	rs_le_z	1	内部	rs 寄存器值是否小于等于 0
20	rs_lt_z	1	内部	rs 寄存器值是否小于 0
21	pc_plus_4	32	内部	PC 加 4，用于计算下一条指令地址
22	br_e	1	内部	分支条件满足标志
23	br_addr	32	内部	分支跳转地址
24	alu_op	12	内部	ALU 操作码，包含加、减、小于、与、或、异或、移位等操作
25	sel_alu_src1	3	内部	ALU 源操作数1 的选择信号
26	sel_alu_src2	4	内部	ALU 源操作数2 的选择信号
27	data_ram_en	1	内部	数据 RAM 使能信号
28	data_ram_wen	4	内部	数据 RAM 写使能信号
29	rf_we	1	内部	寄存器文件写使能信号
30	rf_waddr	5	内部	寄存器文件写地址

31	sel_rf_res	1	内部	寄存器文件结果选择信号
32	sel_rf_dst	3	内部	寄存器文件目标选择信号
33	op_add	1	内部	加法运算指令标志
34	op_sub	1	内部	减法运算指令标志
35	op_slt	1	内部	小于运算指令标志
36	op_sltu	1	内部	无符号小于运算指令标志
37	op_and	1	内部	与运算指令标志
38	op_nor	1	内部	非或运算指令标志
39	op_or	1	内部	或运算指令标志
40	op_xor	1	内部	异或运算指令标志
41	op_sll	1	内部	左移运算指令标志
42	op_srl	1	内部	逻辑右移运算指令标志
43	op_sra	1	内部	算术右移运算指令标志
44	op_lui	1	内部	载入上半字指令标志

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：

序号	接口名	宽度	输入/输出	作用
1	clk	1	输入	时钟信号，驱动模块内部的时序逻辑
2	rst	1	输入	复位信号，高电平有效，用于初始化模块
3	stall	`StallBus`	输入	停顿信号总线，控制流水线的暂停与恢复
4	id_to_ex_bus	`ID_TO_EX_WD`	输入	从 ID 阶段到 EX 阶段的数据总线
5	id_load_bus	`LoadBus`	输入	Load信号总线，指示加载指令类型
6	id_save_bus	`SaveBus`	输入	Save信号总线，指示存储指令类型
7	id_hi_lo_bus	72	输入	从 ID 阶段传递的 HI/LO 寄存器相关信号
8	ex_to_mem_bus	`EX_TO_MEM_WD`	输出	从 EX 阶段到 MEM 阶段的数据总线
9	ex_to_rf_bus	`EX_TO_RF_WD`	输出	从 EX 阶段到寄存器文件的数据总线
10	ex_hi_lo_bus	65	输出	EX 阶段传递给下一阶段的 HI/LO 寄存器相关信号
11	stallreq_for_ex	1	输出	EX阶段发出的停顿请求信号
12	data_sram_en	1	输出	数据SRAM使能信号
13	data_sram_wen	4	输出	数据SRAM写使能信号
14	data_sram_addr	32	输出	数据SRAM地址
15	data_sram_wdata	32	输出	数据SRAM写数据
16	ex_id	1	输出	EX阶段标识信号
17	data_ram_sel	4	输出	数据RAM选择信号
18	ex_load_bus	`LoadBus`	输出	EX阶段Load信号总线，传递具体的Load指令类型

EX 模块详细功能说明

序号	信号名	宽度	输入/输出	作用
1	id_to_ex_bus_r	`ID_TO_EX_WD`	内部	暂存从 ID 阶段传递过来的数据总线
2	id_load_bus_r	`LoadBus`	内部	暂存 Load 信号，总线解码出的具体 Load 指令类型
3	id_save_bus_r	`SaveBus`	内部	暂存 Save 信号，总线解码出的具体 Save 指令类型
4	id_hi_lo_bus_r	72	内部	暂存 HI/LO 寄存器相关信号
5	ex_pc	32	内部	当前指令的 PC 值
6	inst	32	内部	当前指令内容
7	alu_op	12	内部	ALU 操作码，决定 ALU 执行何种运算
8	sel_alu_src1	3	内部	ALU 源操作数1 的选择信号
9	sel_alu_src2	4	内部	ALU 源操作数2 的选择信号
10	data_ram_en	1	内部	数据RAM 使能信号，指示是否进行访存操作
11	data_ram_wen	4	内部	数据RAM 写使能信号，控制写入哪些字节
12	rf_we	1	内部	寄存器文件写使能信号，指示是否写回寄存器
13	rf_waddr	5	内部	寄存器文件写地址，指定写回的寄存器编号
14	sel_rf_res	1	内部	寄存器文件结果选择信号，选择写回数据的来源
15	rf_rdata1	32	内部	寄存器文件读出的数据1
16	rf_rdata2	32	内部	寄存器文件读出的数据2
17	is_in_delayslot	1	内部	标志是否处于延迟槽（分支指令后的指令）
18	byte_sel	4	内部	字节选择信号，用于数据存储
19	imm_sign_extend	32	内部	立即数符号扩展，扩展后的立即数
20	imm_zero_extend	32	内部	立即数零扩展，扩展后的立即数
21	sa_zero_extend	32	内部	位移量零扩展，扩展后的位移量
22	alu_src1	32	内部	ALU 源操作数1
23	alu_src2	32	内部	ALU 源操作数2
24	alu_result	32	内部	ALU 运算结果
25	ex_result	32	内部	EX 阶段的最终结果
26	inst_lb	1	内部	Load 指令类型：加载字节
27	inst_lbu	1	内部	Load 指令类型：加载字节无符号
28	inst_lh	1	内部	Load 指令类型：加载半字
29	inst_lhu	1	内部	Load 指令类型：加载半字无符号
30	inst_lw	1	内部	Load 指令类型：加载字
31	inst_sb	1	内部	Store 指令类型：存储字节

32	inst_sh	1	内部	Store 指令类型：存储半字
33	inst_sw	1	内部	Store 指令类型：存储字
34	inst_mfhi	1	内部	指令：将 HI 寄存器的值移动到通用寄存器
35	inst_mflo	1	内部	指令：将 LO 寄存器的值移动到通用寄存器
36	inst_mthi	1	内部	指令：将通用寄存器的值移动到 HI 寄存器
37	inst_mtlo	1	内部	指令：将通用寄存器的值移动到 LO 寄存器
38	inst_mult	1	内部	指令：有符号乘法
39	inst_multu	1	内部	指令：无符号乘法
40	inst_div	1	内部	指令：有符号除法
41	inst_divu	1	内部	指令：无符号除法
42	hi_we	1	内部	HI 寄存器写使能信号
43	lo_we	1	内部	LO 寄存器写使能信号
44	hi_wdata	32	内部	HI 寄存器写入数据
45	lo_wdata	32	内部	LO 寄存器写入数据
46	mul_result	64	内部	乘法运算结果，64 位
47	mul_signed	1	内部	有符号乘法标记
48	div_result	64	内部	除法运算结果，64 位
49	div_ready_i	1	内部	除法模块准备好信号
50	stallreq_for_div	1	内部	DIV阶段发出的停顿请求信号
51	div_opdata1_o	32	内部	除法操作数1输出寄存器
52	div_opdata2_o	32	内部	除法操作数2输出寄存器
53	div_start_o	1	内部	除法启动信号
54	signed_div_o	1	内部	有符号除法标记

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

序号	接口名	宽度	输入/输出	作用
1	clk	1	输入	时钟信号，驱动模块内部的时序逻辑
2	rst	1	输入	复位信号，高电平有效，用于初始化模块
3	stall	`StallBus`	输入	停顿信号总线，控制流水线的暂停与恢复
4	ex_to_mem_bus	`EX_TO_MEM_WD`	输入	从 EX 阶段传递到 MEM 阶段的数据总线
5	data_sram_rdata	32	输入	从数据 SRAM 读取的数据
6	data_ram_sel	4	输入	数据 RAM 的字节选择信号
7	ex_load_bus	`LoadBus`	输入	从 EX 阶段传递的 Load 指令信号
8	stallreq_for_load	1	输出	对 Load 操作发出的停顿请求信号
9	mem_to_wb_bus	`MEM_TO_WB_WD`	输出	从 MEM 阶段传递到 WB 阶段的数据总线
10	mem_to_rf_bus	`MEM_TO_RF_WD`	输出	从 MEM 阶段传递到寄存器文件的数据总线

MEM 模块详细功能说明

序号	信号名	宽度	输入/输出	作用
1	ex_to_mem_bus_r	`EX_TO_MEM_WD`	内部	暂存从 EX 阶段传递过来的数据总线
2	ex_load_bus_r	`LoadBus`	内部	暂存 Load 指令信号，总线解码出的具体 Load 指令类型
3	data_ram_sel_r	4	内部	暂存数据 RAM 选择信号
4	mem_pc	32	内部	当前指令的 PC 值
5	data_ram_en	1	内部	数据 RAM 使能信号，指示是否进行访存操作
6	data_ram_wen	4	内部	数据 RAM 写使能信号，控制写入哪些字节
7	sel_rf_res	1	内部	寄存器文件结果选择信号，选择写回数据的来源
8	rf_we	1	内部	寄存器文件写使能信号，指示是否写回寄存器
9	rf_waddr	5	内部	寄存器文件写地址，指定写回的寄存器编号
10	rf_wdata	32	内部	寄存器文件写数据，来自访存结果或 EX 阶段运算结果
11	inst_lb	1	内部	Load 指令类型：加载字节
12	inst_lbu	1	内部	Load 指令类型：加载字节无符号
13	inst_lh	1	内部	Load 指令类型：加载半字
14	inst_lhu	1	内部	Load 指令类型：加载半字无符号
15	inst_lw	1	内部	Load 指令类型：加载字
16	inst_sb	1	内部	Store 指令类型：存储字节
17	inst_sh	1	内部	Store 指令类型：存储半字
18	inst_sw	1	内部	Store 指令类型：存储字
19	b_data	8	内部	字节数据，来自数据 SRAM 读取的数据
20	h_data	16	内部	半字数据，来自数据 SRAM 读取的数据
21	w_data	32	内部	字数据，来自数据 SRAM 读取的数据
22	mem_result	32	内部	访存结果，经过扩展处理后的数据
23	rf_wdata	32	内部	寄存器文件写数据，选择访存结果或 EX 阶段结果
24	data_sram_en	1	输出	数据 SRAM 使能信号
25	data_sram_wen	4	输出	数据 SRAM 写使能信号，控制写入哪些字节
26	data_sram_addr	32	输出	数据 SRAM 地址，来自 EX 阶段的运算结果
27	data_sram_wdata	32	输出	数据 SRAM 写数据，来自寄存器文件读取的数据
28	stallreq_for_load	1	输出	对 Load 操作发出的停顿请求信号
29	mem_to_wb_bus	`MEM_TO_WB_WD`	输出	将访存阶段的结果和控制信号打包传递到 WB 阶段
30	mem_to_rf_bus	`MEM_TO_RF_WD`	输出	将访存阶段的结果传递给寄存器文件，用于写回

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

序号	接口名	宽度	输入/输出	作用
1	clk	1	输入	时钟信号，驱动模块内部的时序逻辑
2	rst	1	输入	复位信号，高电平有效，用于初始化模块
3	stall	`StallBus`	输入	停顿信号总线，控制流水线的暂停与恢复
4	mem_to_wb_bus	`MEM_TO_WB_WD`	输入	从 MEM 阶段传递到 WB 阶段的数据总线
5	data_ram_sel	4	输入	数据 RAM 的字节选择信号
6	wb_to_rf_bus	`WB_TO_RF_WD`	输出	从 WB 阶段传递到寄存器文件的数据总线
7	debug_wb_pc	32	输出	调试用：写回阶段的 PC
8	debug_wb_rf_wen	4	输出	调试用：寄存器写使能信号
9	debug_wb_rf_wnum	5	输出	调试用：寄存器写地址
10	debug_wb_rf_wdata	32	输出	调试用：寄存器写数据

WB 模块详细功能说明

序号	信号名	宽度	输入/输出	作用
1	mem_to_wb_bus_r	`MEM_TO_WB_WD`	内部	暂存从 MEM 阶段传递过来的数据总线
2	wb_pc	32	内部	当前指令的 PC 值
3	rf_we	1	内部	寄存器文件写使能信号，指示是否写回寄存器
4	rf_waddr	5	内部	寄存器文件写地址，指定写回的寄存器编号
5	rf_wdata	32	内部	寄存器文件写数据，来自访存结果或 EX 阶段运算结果

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

序号	接口名	宽度	输入/输出	作用
1	rst	1	输入	复位信号，高电平有效，用于初始化模块
2	stallreq_for_ex	1	输入	来自 EX 阶段的停顿请求信号
3	stallreq_for_bru	1	输入	来自分支跳转单元 BRU 的停顿请求信号
4	stallreq_for_load	1	输入	来自 MEM 阶段的 Load 操作的停顿请求信号
5	stall	`StallBus`	输出	停顿信号总线，控制流水线各阶段的暂停与恢复

StallBus 各位含义

序号	StallBus 位	含义
0	stall[0]	取指地址 PC 是否保持不变，为 1 表示保持不变。
1	stall[1]	流水线取指阶段是否暂停，为 1 表示暂停。
2	stall[2]	流水线译码阶段是否暂停，为 1 表示暂停。
3	stall[3]	流水线执行阶段是否暂停，为 1 表示暂停。
4	stall[4]	流水线访存阶段是否暂停，为 1 表示暂停。
5	stall[5]	流水线回写阶段是否暂停，为 1 表示暂停。

> **注**：`StallBus` 的宽度为 6 位，对应 6 个流水线阶段的停顿控制信号。

CTRL 模块详细功能说明

序号	信号名	宽度	输入/输出	作用
1	rst	1	输入	复位信号，高电平有效，用于初始化模块
2	stallreq_for_ex	1	输入	来自 EX 阶段的停顿请求信号
3	stallreq_for_bru	1	输入	来自分支跳转单元 BRU 的停顿请求信号
4	stallreq_for_load	1	输入	来自 MEM 阶段的 Load 操作的停顿请求信号
5	next_inst_rs	5	输入	来自译码阶段的下一指令的源寄存器 rs
6	next_inst_rt	5	输入	来自译码阶段的下一指令的源寄存器 rt
7	current_load_waddr	5	输入	当前 MEM 阶段 Load 指令的目标寄存器地址
8	stall	`StallBus`	输出	停顿信号总线，控制流水线各阶段的暂停与恢复
9	stallreq_for_load_out	1	输出	对 Load 操作发出的停顿请求信号

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

序号	接口名	宽度	输入/输出	作用
1	clk	1	输入	时钟信号，驱动模块内部的时序逻辑
2	stall	`StallBus`	输入	停顿信号总线，用于控制流水线停顿
3	hi_we	1	输入	HI 寄存器写使能信号，指示是否写入 HI 寄存器
4	lo_we	1	输入	LO 寄存器写使能信号，指示是否写入 LO 寄存器
5	hi_wdata	32	输入	HI 寄存器写入数据
6	lo_wdata	32	输入	LO 寄存器写入数据
7	hi_rdata	32	输出	HI 寄存器读出数据
8	lo_rdata	32	输出	LO 寄存器读出数据

HI_LO_REG 模块详细功能说明

序号	信号名	宽度	输入/输出	作用
1	reg_hi	32	内部	内部寄存器，保存 HI 寄存器的当前值
2	reg_lo	32	内部	内部寄存器，保存 LO 寄存器的当前值

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！