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

组长：李天河

组员：董铭扬、希吉尔

报告日期：2024.12.30

目录

一、总体设计概述

1.小组成员工作量划分

姓名	完成任务点	任务量占比
李天河	完成数据相关通路连接，数据相关的解决，暂停的部分实现，完成部分算数和跳转指令	60%
董铭扬	一部分指令的添加、暂停的部分实现	20%
希吉尔	一部分指令的添加、解决部分数据相关	20%

2.总体设计

计算机系统实验中的五级流水线 CPU 设计是一个基于经典 RISC 架构的处理器，实现了指令的高效并行处理。该处理器将指令执行过程分为五个阶段：取指（IF）、译码（ID）、执行（EX）、访存（MEM）和回写（WB）。在每个时钟周期中，指令通过各个阶段流水线的顺序流动，从而实现指令的并行处理。

实验仓库的地址为 https://github.com/Acheron6/computer-architecture。

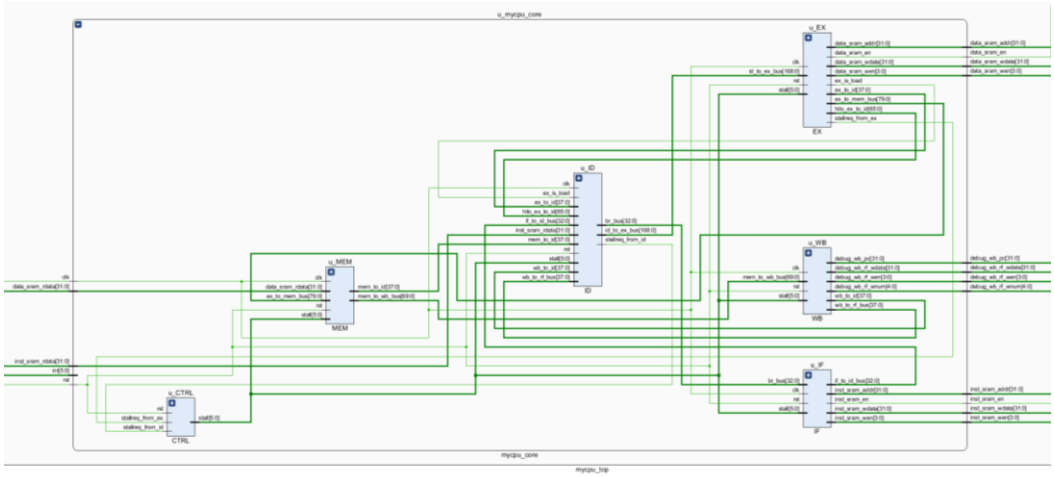
3.实验环境

本实验仿真环境为 vivado2019.2，使用 vscode 编写 verilog 代码并在 vivado 上编译运行代码。

4.流水线简介

流水线是指将计算机指令处理过程拆分为多个步骤，并通过多个硬件处理单 元并行执行来加快指令执行速度。 流水线广泛应用于高档 CPU 的架构中。根据 MIPS 处理器的特点，将整体的 处理过程分为取指令（IF）、指令译码

（ID）、执行（EX）、存储器访问（MEM） 和寄存器回写（WB）五级，对应周期的五个处理阶段。



本实验流水线为五级流水段，分别为 IF、ID、 EX、MEM、WB。 取指阶段：从指令存储器读出指令，同时确定下一条指令地址。 译码阶段：对指令进行译码，从通用寄存器中读出要使用的寄存器的值，如果是转移指令，并且满足转移条件，那么给出转移目标，作为新的指令地址。 执行阶段：按照译码阶段给出的操作数、运算类型，进行运算，给出运算结 果。如果是 load/store 指令，那么还会计算 load/store 的目标地址。访存阶段：如果是 load/store 指令，那么在此阶段回访问数据存储器，反之，只是将执行阶段的结果向下传递到回写阶段。同时，在此阶段还要判断是否有异 常需要处理，如果有，那么回清除流水线，然后转移到异常处理例程入口地址处继续执行。回写阶段：将运算结果保存到目标寄存器。

流水线中经常会发生一些被称为“相关”的情况，它们使得指令序列中的下一条指令无法按照设计的时钟周期执行，这些“相关”会降低流水线的性能。流水线中的相关分为以下三种类型：

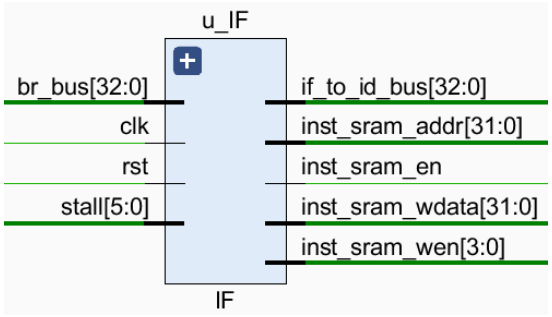
（1）结构相关：指的是在指令执行过程中，由于硬件资源无法满足指令执行的需求，导致硬件资源冲突而产生的相关。例如，指令和数据共享同一个存储器，在某个时钟周期，流水线既需要完成某条指令对存储器中数据的访问操作，又需要执行后续的取指令操作，这样就会发生存储器访问冲突，从而产生结构相关。

（2）数据相关：指的是在流水线中执行的几条指令中，一条指令依赖于前面指令的执行结果。这种相关通常会导致指令的延迟，直到前面的指令完成其数据计算。

（3）控制相关：指的是流水线中的分支指令或其他需要修改程序计数器（PC）的指令所造成的相关。分支指令会改变程序的执行路径，导致后续指令的执行顺序发生变化，进而影响流水线的正常执行。

二、流水线各个阶段说明

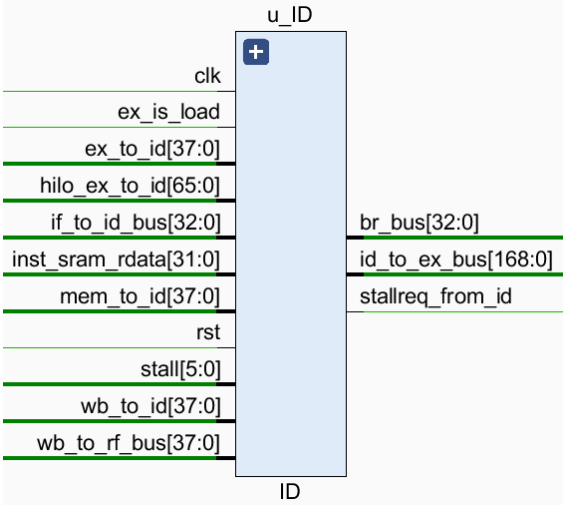
1.取值(IF)段



序号	接口名宽度	宽度	输入/输出	作用
1	clk	1	输入	时钟信号
2	rst	1	输入	复位信号
3	br_bus	33	输入	控制暂停信号
4	stall	6	输入	ID 段分支跳转指令信号
5	if_to_id_bus	33	输出	IF 传给 ID 的数据
6	inst_sram_en	1	输出	指令寄存器读写使能信号
7	inst_sram_wen	4	输出	指令寄存器写使能信号
8	inst_sram_addr	32	输出	指令寄存器地址
9	inst_sram_wdata	32	输出	指令寄存器写数据

IF 阶段接收时钟信号和复位信号，当复位信号为有效时，PC 值将被设置为复位地址。输入的暂停信号由控制模块发出，作用是当流水线暂停时，IF 阶段根据 stall 信号暂停指令的延迟执行，确保下一条指令的 PC 值与当前的 PC 值一致，从而使 IF 阶段暂停一个时钟周期。br_bus 信号由 ID 阶段发出，包含跳转信息，包括是否跳转以及跳转的目标地址。如果 IF 阶段检测到跳转请求，会将 next_pc 更新为跳转地址；如果没有暂停或跳转，reg_pc 将被更新为当前的 next_pc 值，且 next_pc 会增加 4，指向下一条指令。最后，reg_pc 的值作为地址传递给指令存储器，从中读取对应地址的指令，并将其传递给 ID 阶段。

2.译码(ID)段



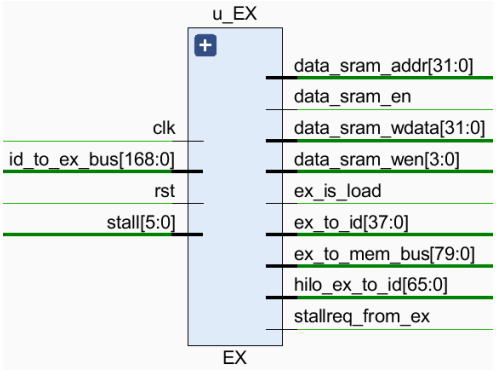
序号	接口名	宽度	输入/输出	作用
1	clk	1	输入	时钟信号
2	rst	1	输入	复位信号
3	stall	6	输入	控制暂停信号
4	ex_is_load	1	输入	指令是否加载
5	if_to_id_bus	33	输入	IF 传给 ID 的数据
6	wb_to_rf_bus	38	输入	WB 传给 ID 段的值
7	inst_sram_rdata	32	输入	当前指令地址存储的值

8	wb_to_id	38	输入	WB 传给 ID 的数据
9	mem_to_id	38	输入	MEM 传给 ID 的数据
10	ex_to_id	38	输入	EX 传给 ID 的数据
11	hilo_ex_to_id	66	输入	EX 传 ID 写入乘除法寄存器的值
12	br_bus	33	输出	ID 传给 IF 的数据
13	stallreq_from_id	1	输出	ID 发出的暂停信号
14	id_to_ex_bus	169	输出	ID 传给 EX 的据
15	stallreq	1	输出	控制暂停

ID 阶段在处理器的流水线中负责将从 IF 阶段获取到的指令进行解码，并准备好后续执行所需的操作数和控制信号。在此过程中，ID 阶段需要判断指令类型，读取寄存器文件中的数据，处理跳转指令，检查是否存在数据依赖以及决定是否需要暂停流水线执行。当指令涉及到加载、存储或者运算时，ID 阶段还会进行数据转发，避免因数据未准备好导致错误执行。如果流水线中的其他阶段（如 EX 阶段）尚未完成数据写回，ID 阶段可能会发出暂停信号，等待数据准备完毕。此外，ID 阶段还需要根据跳转指令的类型，判断是否发生跳转，并通过信号控制程序计数器（PC）的更新。

在处理数据依赖和指令冲突时，ID 阶段的作用尤为重要。如果当前指令依赖于前一个阶段的数据结果，ID 阶段会发出暂停请求，确保数据依赖问题得到解决，避免错误的计算或访问。通过与 EX、MEM、WB 阶段的交互，ID 阶段确保指令能够顺利执行，且数据的流动和控制逻辑能够正确无误地传递。同时，ID 阶段还要处理高低位寄存器的数据转发，尤其是在乘法、除法等操作中。

3.执行(EX)段



序号	接口名	宽度	输入/输出	作用
1	clk	1	输入	时钟信号
2	rst	1	输入	复位信号
3	stall	6	输入	控制暂停信号
4	id_to_ex_bus	169	输入	ID 传给 EX 的数据
5	data_sram_wen	4	输出	内存数据写使能信号
6	data_sram_addr	32	输出	内存数据存放地址
7	data_sram_en	1	输出	内存数据读写使能信号
8	ex_to_id	38	输出	EX 传给 ID 数据
9	ex_to_mem_bus	80	输出	EX 传给 MEM 数据
10	hilo_ex_to_id	66	输出	EX 将乘除法器的结果传给 ID
11	ex_is_load	1	输出	判断指令是否是存储指令
12	stallreq_from_ex	1	输出	EX 传出的是否暂停信号
13	data_sram_wdata	32	输出	写入内存的数据

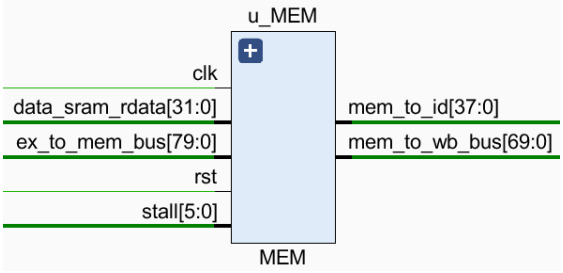
判断当前指令是否为 LW 指令 ：检查指令的前 6 位（inst[31:26]）是否为 6'b10_0011。如果是，则将 ex_is_load 赋值为 1；如果不是，则将 ex_is_load 赋值为 0。计算 ALU 操作数的值。首先，定义一个立即数符号扩展变量，并将

其值设置为{{16{inst[15]}}, inst[15:0]}；定义一个立即数零扩展变量，并将其值设置为{16'b0, inst[15:0]}；定义一个 sa 零扩展变量，并将其值设置为{27'b0, inst[10:6]}。根据 ID 阶段传递的操作数来源方式，即 sel_alu_src1 和 sel_alu_src2 的值，确定参与 ALU 运算的操作数，并将其分别赋值给 alu_src1 和 alu_src2。调用 ALU 模块，将两个操作数和 ALU 计算方式传递给 ALU 接口，并从 ALU 模块中获取计算结果，将其赋值给 ex_result。

读内存：在设置 data_sram_en 为相应值后，1 表示可以读取内存，0 表示无法读取内存。同时，data_sram_addr 被赋值为要访问的内存地址，EX 阶段将获取该地址的数据。

发送 EX 阶段结果给其他阶段：发送给 WB 阶段：包括内存的读使能信号、当前 PC 值、内存的读写使能信号、内存写使能信号、寄存器写使能信号，以及要写入寄存器的地址和数据。发送给 ID 阶段：包括寄存器的写使能信号、寄存器的写入地址和数据，用于 ID 阶段判断是否会发生相关情况。另外，还包括乘除法器的高位和低位寄存器的写使能信号，以及相应的数据，帮助 ID 阶段在操作 regfile 时同步将乘除法器的高低位数据写入寄存器，从而提升 CPU 的整体效率。

4.访存(MEM)段



确定写入寄存器的值：该值可能来自 ALU 计算结果，也可能是访存阶段从内存读取的值。首先判断是否是 lw 指令，如果是，则从内存读取的值写入寄存器；如果是 lb 指令，根据地址的后两位决定从内存读取的字节并进行符号扩展；若是 lbu 指令，则进行零扩展；对于 lh 指令，根据地址后两位决定读取的两个字节并进行符号扩展；若是 lhu 指令，则进行零扩展。若都不是，使用 ALU 计算结果作为写入值。

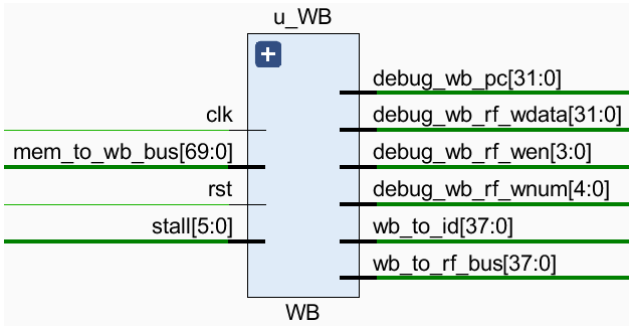
传值给 WB 阶段：将当前的 PC 值，以及是否要写入寄存器，以及要写入寄存器的值还有数据发给回写段。

传值给 ID 段：将当前的 PC 值，以及是否要写入寄存器，以及要写入寄存器的值还有数据发给译码段，用来判断是否发生数据相关。

```
assign ex_to_mem_bus = {
        data_ram_readen,//79:76
        ex_pc,          // 75:44
        data_ram_en,    // 43
        data_ram_wen,   // 42:39
        sel_rf_res,     // 38
        rf_we,          // 37
        rf_waddr,       // 36:32
        ex_result       // 31:0
    };
```

这段代码的功能是将多个信号按指定顺序打包成一个总线 ex_to_mem_bus，以便在执行阶段（EX）和访存阶段（MEM）之间传递数据。具体而言，它将数据存储器的读使能信号 data_ram_readen、执行阶段的程序计数器 ex_pc、数据存储器的使能信号 data_ram_en、写使能信号 data_ram_wen、选择寄存器文件写入结果源的信号 sel_rf_res、寄存器文件的写使能信号 rf_we、寄存器写入地址 rf_waddr 和执行阶段计算结果 ex_result 打包在一起，形成一个包含所有控制信息和数据的总线，从而实现跨阶段的数据传递。

5.回写(WB)段

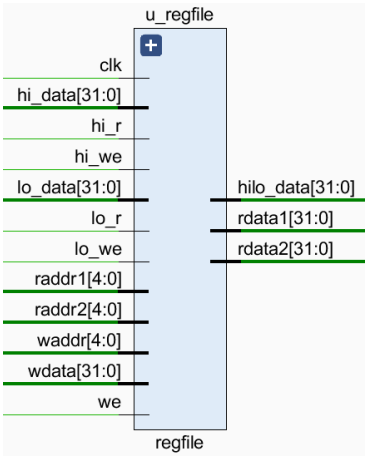


序号	接口名	宽度	输入/输出	作用

1	clk	1	输入	时钟信号
2	rst	1	输入	复位信号
3	stall	6	输入	控制暂停信号
4	mem_to_wb_bus	70	输入	MEM 传给 WB 的数据
5	debug_wb_rf_wen	4	输出	用来 debug 的写使能信号
6	debug_wb_rf_wnum	5	输出	用来 debug 的写寄存器地址
7	debug_wb_rf_wdata	32	输出	用来 debug 的写寄存器数据
8	debug_wb_pc	32	输出	用来 debug 的 pc 值
9	wb_to_rf_bus	38	输出	WB 传给 RF 的数据
10	wb_to_id	38	输出	WB 传给 ID 的数据

模块中实现的 MEM 模块的输出 rf_we、rf_waddr、rf_wdata 连接到 Regfile 模块,分别连接到写使能端口 we、写操作目的寄存器端口 waddr、写入数据端口 wdata,所以会将指令的运算结果写入目的寄存器。

6.Regfile 寄存器



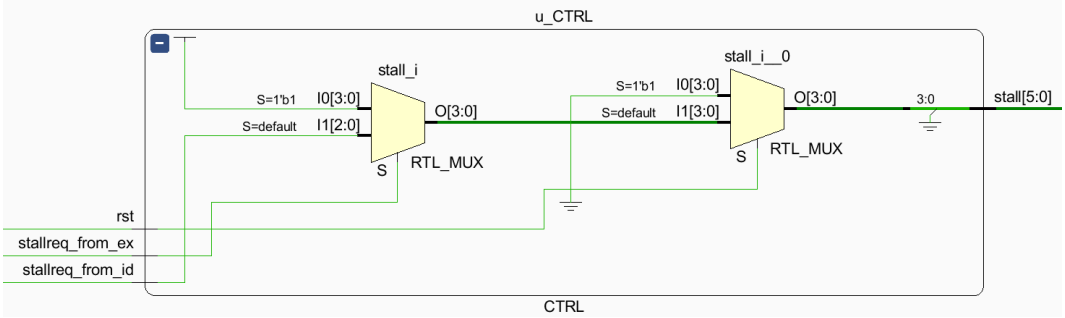
序号	接口名	宽度	输入/输出	作用
1	clk	1	输入	时钟信号

2	raddr1	5	输入	读第一个数的地址
3	raddr2	5	输入	读第二个数的地址
4	we	1	输入	寄存器写使能信号
5	waddr	5	输入	写地址
6	wdata	32	输入	写寄存器的值
7	hi_r	1	输入	hi 寄存器读使能信号
8	hi_we	1	输入	hi 寄存器写使能信号
9	hi_data	32	输入	写 hi 寄存器的值
10	lo_r	1	输入	lo 寄存器读使能信号
11	lo_we	1	输入	lo 寄存器写使能信号
12	lo_data	32	输入	写 lo 寄存器的值
13	hilo_data	32	输出	hilo 寄存器读出来的值
14	rdata1	32	输出	读出第一个数的值
15	rdata2	32	输出	读出第二个数的值

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。（在本实验中目前为止只有两个会发送请求信号，译码阶段和 执行阶段，取值和访存阶段没有暂停请求，应为这列个阶段的操作都可以在一个周期内完成）

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、第 n 阶段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。

7.CTRL 模块



序号	接口名	宽度	输入/输出	作用
1	rst	1	输入	复位信号
2	stall	6	输出	暂停流水线控制信号
3	stallreq_from_ex	1	输入	EX 段是否请求流水线暂停
4	stallreq_from_id	1	输入	ID 段是否请求流水线暂停

这个模块用于控制流水线中的暂停机制，确保处理器能够正确处理数据冒险或控制冒险等问题。它通过接收来自不同阶段的暂停请求信号（如执行阶段和指令解码阶段），来决定是否暂停流水线中的某些阶段。具体来说，当 stallreq_from_ex（来自执行阶段的暂停请求）为高电平时，表示执行阶段遇到问题需要暂停，此时模块会产生暂停信号，通常是暂停除 IF（指令获取）阶段外的所有阶段；当 stallreq_from_id（来自指令解码阶段的暂停请求）为高电平时，表示指令解码阶段无法继续，模块会产生暂停信号，通常是暂停从 EX（执行）到 WB（写回）阶段。这些信号通过控制 stall 信号来实现，其中 stall 是一个多位信号，每一位对应一个流水线阶段的暂停状态。当系统复位时，所有的暂停信号被清除，流水线恢复正常执行。通过这种机制，模块能够根据不同的暂停请求动态调整流水线的执行状态，防止因数据依赖或其他问题导致错误的执行结果，保证指令的正确执行和数据的一致性。

六、小组各成员的收获与感受

1.李天河

通过实验，我深刻体会到流水线结构不仅能提升 CPU 的吞吐量，还能增加设计的复杂性。在每个流水线阶段的设计中，我需要权衡性能与实现的难度，力求使整个流水线的各个部分协调工作，达到最优化。尤其是在模拟实现时，如何确保各个流水线寄存器的正确传输和同步操作，避免由于时序问题而导致的错误，是一项具有挑战性的工作。实验也让我更加重视硬件与软件之间的紧密配合。在设计流水线时，如何与编译器协同工作，减少指令调度时的冲突，提升流水线的效率，是现代 CPU 设计中一个不可忽视的环节。

除了收获，在进行实验的过程中，我还遇到了很多困难！对新的编程软件不了解，看不懂给的提示等等。在同学和学长的帮助下，我慢慢的解决了这个问题，逐渐完成了新的代码。通过这次实验，我也了解了我的不足之处，在之后的学习中会更加的努力，对计算机系统的知识进行更详细的了解。

2.董铭扬

通过进行计算机系统的实验，我深入了解了计算机是如何从硬件到软件逐步实现工作的。特别是在操作系统、存储管理、进程调度、内存管理等方面的知识让我对计算机的底层机制有了更深的认识。曾经只是在应用层做程序开发，但了解了操作系统如何管理硬件资源、如何调度进程等内容后，才更加明白应用程序与操作系统之间的互动关系。

3.希吉尔

在设计过程中，我不仅需要理解每个阶段的功能，还需要考虑如何避免流水线中的数据冲突、控制冲突和结构冲突。例如，当一个指令在执行阶段需要使用前一个指令的结果时，必须设计有效的数据转发机制或插入气泡来处理数据依赖问题。控制冲突的解决方法，如分支预测，也是设计中需要特别注意的方面。

参考文献：

[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

[3] “龙芯杯”计算机系统能力大赛参考资料