成绩评定日期：

实验报告成绩：

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2201

组长：张一航

组员：王梓豪 雷鹏

报告日期：2024.12.11

目录

[1 实验概述 3](#_Toc15703)

[1.1小组成员及任务分工 3](#_Toc8633)

[1.2任务概述 3](#_Toc17298)

[1.3运行环境 4](#_Toc16245)

[2 理论基础 4](#_Toc31796)

[2.1 五级流水线CPU结构 4](#_Toc30385)

[2.2 数据冒险与控制冒险 5](#_Toc28478)

[2.3 流水线优化技术 5](#_Toc7927)

[3 实验方法与流程 6](#_Toc3934)

[3.1 设计流程 6](#_Toc7267)

[3.2 模块划分 6](#_Toc984)

[3.3 接口定义 7](#_Toc26199)

[4 代码说明 8](#_Toc26357)

[4.1 IF段 8](#_Toc21695)

[4.2 ID段 9](#_Toc17572)

[4.3EX段 14](#_Toc28826)

[4.4 MEM段 16](#_Toc17268)

[4.5 WB段 17](#_Toc10659)

[4.6 STALL CTRL段 18](#_Toc29201)

[5 心得体会 20](#_Toc8773)

[5.1张一航的心得体会 20](#_Toc17892)

[5.2 王梓豪的心得体会 20](#_Toc3143)

[5.3雷鹏的心得体会 21](#_Toc31630)

[结论 22](#_Toc9176)

# 1 实验概述

## **1.1**小组成员及任务分工

|  |  |  |
| --- | --- | --- |
| 姓名 | 任务分工 | 任务量占比 |
| 张一航 | 参与ID段设计，完成EX、MEM阶段数据通路连接，完成store、load指令的部分逻辑 | 40% |
| 王梓豪 | 主要负责EX和MEM模块，实现高效准确的ALU操作和地址计算。集成并优化乘法和除法模块，确保复杂运算的正确性和效率。 | 40% |
| 雷鹏 | 主要负责WB和CRTL模块 | 20% |

## **1.2**任务概述

本次实验要求完成一个五级流水线的CPU设计。具体任务包括在现有框架代码的基础上，补充逻辑指令、算数指令、跳转指令及数据相关的处理。主要模块包括：

IF.v：取指阶段

ID.v：译码阶段

EX.v：执行阶段

MEM.v：访存阶段

WB.v：回写阶段

CTRL.v：流水线控制模块，负责暂停和恢复流水线

此外，lib文件夹下提供了必要的库文件，CTRL文件用于控制流水线的暂停和继续操作。图示

描述已自动生成

## **1.3**运行环境

· 硬件环境：采用CG服务器进行开发和仿真。

· 软件环境：Vivado 2019版本，用于Verilog代码的编写、综合和仿真。

# 2 理论基础

## 2.1 五级流水线CPU结构

五级流水线是经典的RISC（精简指令集计算机）处理器设计，通过将指令执行过程划分为五个独立的阶段，以实现指令的并行处理。这五个阶段分别是：

· 取指阶段（IF，Instruction Fetch）：从指令存储器中取出指令，并计算下一条指令的地址。

· 译码阶段（ID，Instruction Decode）：对取出的指令进行译码，读取操作数，并进行必要的寄存器重命名和转发。

· 执行阶段（EX，Execution）：执行算术或逻辑运算，计算分支地址或内存地址。

· 访存阶段（MEM，Memory Access）：访问数据存储器，进行读写操作。

· 回写阶段（WB，Write Back）：将运算结果写回寄存器。

通过将这五个阶段并行化，不同指令可以在不同的阶段同时进行处理，从而提高处理器的整体性能。

## 2.2 数据冒险与控制冒险

**数据冒险**

数据冒险是指指令之间存在数据依赖关系，导致流水线无法正确执行的情况。主要分为三种类型：

· 读后写（RAW，Read After Write）：后续指令需要读取前一指令写入的寄存器，但写入操作尚未完成。

· 写后读（WAR，Write After Read）：后续指令需要写入一个寄存器，而前一指令尚未读取该寄存器。

· 写后写（WAW，Write After Write）：后续指令需要写入一个寄存器，而前一指令尚未完成写入操作。

为解决数据冒险问题，常用的方法包括数据前递（数据转发）和流水线暂停（插入气泡）。

**控制冒险**

控制冒险主要发生在分支指令执行时，处理器需要决定下一条指令的地址。如果分支预测错误，可能导致流水线中的指令需要被清除或重新取指，从而影响性能。

解决控制冒险的方法包括：

· 静态分支预测：基于固定策略预测分支方向，如总是假设分支不跳转。

· 动态分支预测：根据历史执行情况预测分支方向，通常通过分支预测器实现。

· 延迟分支：将分支指令后的指令调整为延迟槽指令，以减少流水线冲突。

## 2.3 流水线优化技术

为了进一步提升流水线的性能，可以采用以下优化技术：

· 分支预测：通过预测分支指令的跳转方向，减少控制冒险带来的流水线停顿。

· 超标量技术：在每个时钟周期内发射多条指令，提高指令吞吐量。

· 动态调度：通过乱序执行和指令重排，提高指令级并行度。

· 缓存技术：利用指令缓存和数据缓存减少内存访问延迟。

通过这些优化技术，可以显著提高流水线CPU的执行效率和整体性能。

# 3 实验方法与流程

## 3.1 设计流程

本次实验的设计流程主要包括以下几个步骤：

· 需求分析：明确实验目标，了解五级流水线CPU的基本结构和功能需求。

· 模块设计：根据流水线阶段划分模块，设计各个模块的功能和接口。

· 代码编写：使用Verilog HDL编写各个模块的代码，实现指令的取指、译码、执行、访存和回写功能。

· 模块集成：将各个模块集成到整体CPU架构中，确保模块间的正确连接和数据传输。

· 仿真测试：使用Vivado进行仿真，设计测试用例，验证CPU的功能正确性。

· 调试优化：根据仿真结果，定位和修复设计中的问题，优化性能。

· 文档撰写：整理实验过程和结果，编写实验报告。

## 3.2 模块划分

根据五级流水线的结构，将CPU划分为以下主要模块：

· IF段（取指模块）：负责从指令存储器中取出指令，并计算下一条指令的地址。

· ID段（译码模块）：对取出的指令进行译码，读取操作数，并生成控制信号。

· EX段（执行模块）：执行算术或逻辑运算，计算分支地址或内存地址。

· MEM段（访存模块）：访问数据存储器，进行读写操作。

· WB段（回写模块）：将运算结果写回寄存器。

· CTRL段（控制模块）：负责流水线的暂停和恢复，处理数据冒险和控制冒险。

每个模块之间通过数据总线进行通信，确保指令和数据的正确传输。

## 3.3 接口定义

为了确保模块之间的正确连接和数据传输，需要详细定义各个模块的接口信号。以下是各模块的主要接口定义：

**IF段接口：**

· 输入：时钟信号（clk）、复位信号（rst）、暂停信号（stall）、分支跳转· 信号（br\_bus）

· 输出：IF段到ID段的数据总线（if\_to\_id\_bus）、指令存储器使能信号（inst\_sram\_en）、写使能信号（inst\_sram\_wen）、指令地址（inst\_sram\_addr）、指令数据（inst\_sram\_wdata）

**ID段接口：**

· 输入：时钟信号（clk）、复位信号（rst）、暂停信号（stall）、IF段数据（if\_to\_id\_bus）、指令数据（inst\_sram\_rdata）、来自EX段的数据（ex\_id）、来自WB段的数据（wb\_to\_rf\_bus）、来自EX段的寄存器数据（ex\_to\_rf\_bus）、来自MEM段的寄存器数据（mem\_to\_rf\_bus）、来自EX段的hilo寄存器数据（ex\_hi\_lo\_bus）

· 输出：ID段到EX段的数据总线（id\_to\_ex\_bus）、分支跳转信号（br\_bus）、ID段的hilo寄存器数据（id\_hi\_lo\_bus）、执行load命令的数据总线（id\_load\_bus）、执行save命令的数据总线（id\_save\_bus）、暂停请求信号（stallreq\_for\_bru）

**EX段接口：**

· 输入：时钟信号（clk）、复位信号（rst）、暂停信号（stall）、ID段数据（id\_to\_ex\_bus）、ID段的load数据（id\_load\_bus）、ID段的save数据（id\_save\_bus）

· 输出：EX段到MEM段的数据（ex\_to\_mem\_bus）、EX段到寄存器文件的数据（ex\_to\_rf\_bus）、EX段到hilo寄存器的数据（ex\_hi\_lo\_bus）、对EX段的stall请求（stallreq\_for\_ex）、内存数据的读写使能信号（data\_sram\_en）、内存数据的写使能信号（data\_sram\_wen）、内存数据地址（data\_sram\_addr）、要写入内存的数据（data\_sram\_wdata）、EX段传给ID段的数据（ex\_id）、内存数据选择信号（data\_ram\_sel）、EX段读取的数据（ex\_load\_bus）

**MEM段接口：**

· 输入：时钟信号（clk）、复位信号（rst）、暂停信号（stall）、EX段数据（ex\_to\_mem\_bus）、从内存中读出的数据（data\_sram\_rdata）、内存数据选择信号（data\_ram\_sel）、EX段读取的数据（ex\_load\_bus）

· 输出：MEM段到WB段的数据（mem\_to\_wb\_bus）、MEM段到寄存器文件的数据（mem\_to\_rf\_bus）、对EX段的stall请求（stallreq\_for\_load）

**WB段接口：**

· 输入：时钟信号（clk）、复位信号（rst）、暂停信号（stall）、MEM段数据（mem\_to\_wb\_bus）

· 输出：WB段传给寄存器文件的数据（wb\_to\_rf\_bus）、用于调试的PC值（debug\_wb\_pc）、用于调试的写使能信号（debug\_wb\_rf\_wen）、用于调试的写寄存器地址（debug\_wb\_rf\_wnum）、用于调试的写寄存器数据（debug\_wb\_rf\_wdata）

**CTRL段接口：**

· 输入：时钟信号（clk）、EX段的stall请求（stallreq\_for\_ex）、BRU的stall请求（stallreq\_for\_bru）、Load命令的stall请求（stallreq\_for\_load）

· 输出：暂停信号（stall）

通过详细定义各模块的接口，可以确保模块之间的数据传输和控制信号的正确连接，从而实现流水线CPU的功能。

# 4 代码说明

## **4.1** IF段

该段的输入输出总线定义如下方图片及表格所示：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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 | 暂停信号，控制指令是否暂停 |
| 4 | br\_bus | 33 | 输入 | 分支跳转信号，控制延迟槽是否跳转 |
| 5 | if\_to\_id\_bus | 33 | 输出 | IF段到ID段的数据总线 |
| 6 | inst\_sram\_en  inst\_sram\_en  \_en | 1 | 输出 | 读写使能信号 |
| 7 | inst\_sram\_wen | 4 | 输出 | 写使能信号 |
| 8 | inst\_sram\_addr | 32 | 输出 | 存放指令寄存器的地址 |
| 9 | inst\_sram\_wdata | 32 | 输出 | 存放指令寄存器的数据 |

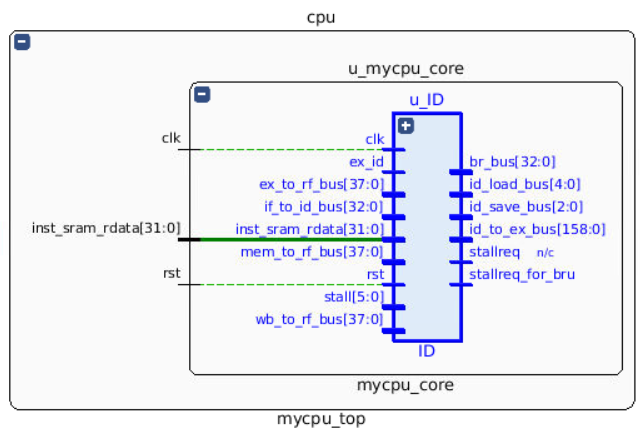
图示

描述已自动生成

这一阶段相对简单，没有太多需要改动的地方，大部分时候负责的工作是从PC+4的地址处获取下一条指令。只需要注意几个跳转指令需要跳到对应的地址读取指令。

## **4.2** ID段

该段的输入输出总线定义如下方图片及表格所示：



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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 | 暂停信号，控制指令是否暂停 |
| 4 | stallreq | 1 | 输出 | 暂停请求信号 |
| 5 | if\_to\_id\_bus | 33 | 输入 | IF段到ID段的数据总线 |
| 6 | inst\_sram\_rdata | 1 | 输入 | 读写使能信号 |
| 7 | ex\_id | 1 | 输入 | 写使能信号 |
| 8 | wb\_to\_rf\_bus | 38 | 输入 | WB段存放进寄存器的数据 |
| 9 | ex\_to\_rf\_bus | 38 | 输入 | EX段存放进寄存器的数据 |
| 10 | mem\_to\_rf\_bus | 38 | 输入 | MEM段存放进寄存器的数据 |
| 11 | ex\_hi\_lo\_bus | 66 | 输入 | EX段存放进hilo寄存器的数据的总线 |
| 12 | id\_hi\_lo\_bus | 72 | 输出 | ID段存放进hilo寄存器的数据的总线 |
| 13 | id\_load\_bus | 5 | 输出 | ID段执行load命令的数据总线 |
| 14 | id\_save\_bus | 3 | 输出 | ID段执行save命令的数据总线 |
| 15 | stallreq\_for\_bru | 1 | 输出 | 执行load命令时的暂停请求 |
| 16 | id\_to\_ex\_bus | 159 | 输出 | ID段到EX段的数据总线 |
| 17 | br\_bus | 33 | 输出 | 分支跳转信号，控制延迟槽是否跳转 |

ID段的主要功能是指令解码。

日程表

描述已自动生成

根据32位MIPS指令集类型可获取译码结果。根据I、J、R型指令对应部分读取寄存器文件中地址为 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。（由于数据相关，在这里，传给下一段的寄存器数据是ndata而非rdata）

    assign ndata1 = ((ex\_rf\_we && rs == ex\_rf\_waddr) ? ex\_rf\_wdata : 32'b0) |

                   ((!(ex\_rf\_we && rs == ex\_rf\_waddr) && (mem\_rf\_we && rs == mem\_rf\_waddr)) ? mem\_rf\_wdata : 32'b0) |

                   ((!(ex\_rf\_we && rs == ex\_rf\_waddr) && !(mem\_rf\_we && rs == mem\_rf\_waddr) && (wb\_rf\_we && rs == wb\_rf\_waddr)) ? wb\_rf\_wdata : 32'b0) |

                   (((ex\_rf\_we && rs == ex\_rf\_waddr) || (mem\_rf\_we && rs == mem\_rf\_waddr) || (wb\_rf\_we && rs == wb\_rf\_waddr)) ? 32'b0 : rdata1);

    assign ndata2 = ((ex\_rf\_we && rt == ex\_rf\_waddr) ? ex\_rf\_wdata : 32'b0) |

                   ((!(ex\_rf\_we && rt == ex\_rf\_waddr) && (mem\_rf\_we && rt == mem\_rf\_waddr)) ? mem\_rf\_wdata : 32'b0) |

                   ((!(ex\_rf\_we && rt == ex\_rf\_waddr) && !(mem\_rf\_we && rt == mem\_rf\_waddr) && (wb\_rf\_we && rt == wb\_rf\_waddr)) ? wb\_rf\_wdata : 32'b0) |

                   (((ex\_rf\_we && rt == ex\_rf\_waddr) || (mem\_rf\_we && rt == mem\_rf\_waddr) || (wb\_rf\_we && rt == wb\_rf\_waddr)) ? 32'b0 : rdata2);

同时根据opcode和func部分分析要执行的运算，给对应的ALU的相关控制逻辑赋值，其中，0表示该条指令不采用该 ALU，1 表示采用该ALU，同时将所有的 ALU 标识符组合起来成为 alu\_op，alu\_op为十二位宽，代表 16 种不同的 ALU，并且作为传入 EX 段的一部分。要写入的目的寄存器。rf\_we 代表写使能信号，表示该条指令是否用写入通用寄存器，sel\_rf\_dst[0]表示该指令要将计算结果写入 rd 通用寄存器，sel\_rf\_dst[1]表示该指令要将计算结果写入 rt 通用寄存器,sel\_rf\_dst[2]表示该指令要将计算结果写入 31 号通用寄存器。rf\_waddr 表示要该条指令的计算结果要写入的通用寄存器的地址，data\_ram\_en表示该条指令是否要与内存中取值或者写入值，如果该条指令要从内存中取值或者写入值，那么它将被赋值为 1’b1,data\_ram\_wen 为四位宽，表示该条指令是否要写入寄存器，如果该条指令要将计算结果的第几个字节写入寄存器，那么对应位置的值设为 1。

用sel\_alu\_src1和sel\_alu\_src2来判断操作数来源，第一个操作数有三种来源，第二个操作数有四种来源，通过区分不同的指令进而分辨不同的操作数的来源。

传值用ID段得到的数据，分别给id\_to\_ex\_bus和br\_bus赋值，其中br\_bus是传给IF段的用于传输跳转指令的判断信号和跳转的地址。

    // rs to reg1

    assign sel\_alu\_src1[0] =    inst\_lw | inst\_sw |inst\_lb| inst\_lbu  | inst\_lh | inst\_lhu| inst\_sb | inst\_sh |

                                inst\_ori | inst\_addiu | inst\_or | inst\_xor | inst\_and  | inst\_andi| inst\_nor | inst\_xori |

                                inst\_sub | inst\_subu | inst\_add | inst\_addi | inst\_addu |

                                inst\_jr | inst\_bgezal | inst\_bltzal |

                                inst\_slti | inst\_or | inst\_srav | inst\_sltu | inst\_slt | inst\_sltiu | inst\_sllv| inst\_srlv

                                ;

    // pc to reg1

    assign sel\_alu\_src1[1] = inst\_jal | inst\_jalr | inst\_bltzal | inst\_bgezal;

    // sa\_zero\_extend to reg1

    assign sel\_alu\_src1[2] = inst\_sll | inst\_sra | inst\_srl;

    // rt to reg2

    assign sel\_alu\_src2[0] =    inst\_sub | inst\_subu | inst\_addu | inst\_sll | inst\_or | inst\_xor | inst\_sra | inst\_srl |

                                inst\_srlv | inst\_sllv| inst\_sra | inst\_srav| inst\_sltu | inst\_slt  | inst\_add | inst\_and| inst\_nor ;

                                // inst\_div | inst\_divu |inst\_mult | inst\_multu;

    // imm\_sign\_extend to reg2

    assign sel\_alu\_src2[1] =    inst\_lui | inst\_addiu | inst\_lw | inst\_sw | inst\_slti| inst\_sltiu | inst\_addi |

                                inst\_lb  | inst\_lbu   | inst\_lh  | inst\_lhu | inst\_sh | inst\_sb;

    // 32'b8 to reg2

    assign sel\_alu\_src2[2] = inst\_jal | inst\_jalr | inst\_bgezal | inst\_bltzal;

    // imm\_zero\_extend to reg2

    assign sel\_alu\_src2[3] = inst\_ori | inst\_andi | inst\_xori;

跳转指令：先用br\_e表示这条指令是否为跳转指令，用rs\_ge\_z表示是否满足 rdata1的值大于等于 0，用rs\_le\_z表示是否满足rdata1的值小于等于0，用rs\_lt\_z表示是否满足 rdata1的值小于 0，rs\_eq\_rt表示是否满足rdata1是否等于radta2的值。br\_addr 表示跳转后的地址，根据不同的指令对地址做不同的计算，并将结果赋给br\_addr。

    assign br\_e = inst\_beq & rs\_eq\_rt | inst\_j | inst\_jalr | | inst\_jr | inst\_jal | inst\_bne & ~rs\_eq\_rt |

                  inst\_bgez & rs\_ge\_z | inst\_bgtz & rs\_gt\_z |inst\_blez & rs\_le\_z | inst\_bltz & rs\_lt\_z |

                  inst\_bgezal & rs\_ge\_z | inst\_bltzal & rs\_lt\_z ;

    assign br\_addr =

                        (inst\_beq       ? (pc\_plus\_4 + {{14{inst[15]}},inst[15:0],2'b0}) : 32'b0)   |

                        (inst\_jr        ? ndata1 : 32'b0)                                           |

                        (inst\_jal       ? {pc\_plus\_4[31:28],instr\_index,2'b0} : 32'b0)              |

                        (inst\_j         ? ({ pc\_plus\_4[31:28],inst[25:0],2'b0}) : 32'b0)            |

                        (inst\_jalr      ? ndata1:32'b0)                                             |

                        (inst\_bne       ? (pc\_plus\_4 + {{14{inst[15]}},inst[15:0],2'b0}) : 32'b0)   |

                        (inst\_bgez      ? (pc\_plus\_4 + {{14{inst[15]}},inst[15:0],2'b0}) : 32'b0)   |

                        (inst\_bgtz      ? (pc\_plus\_4 + {{14{inst[15]}},inst[15:0],2'b0}) : 32'b0)   |

                        (inst\_blez      ? (pc\_plus\_4 + {{14{inst[15]}},inst[15:0],2'b0}) : 32'b0)   |

                        (inst\_bltz      ? (pc\_plus\_4 + {{14{inst[15]}},inst[15:0],2'b0}) : 32'b0)   |

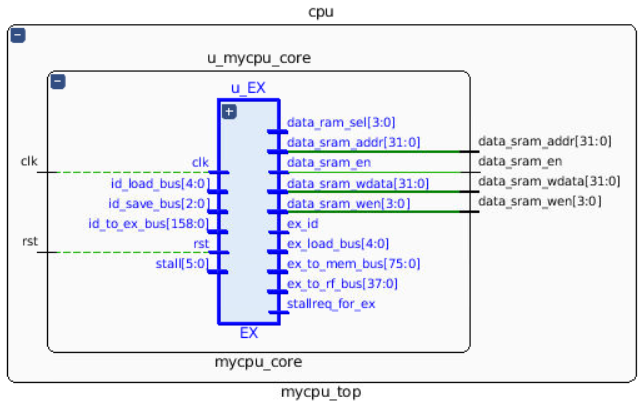
                        (inst\_bgezal    ? (pc\_plus\_4 + {{14{inst[15]}},inst[15:0],2'b0}) : 32'b0)   |

                        (inst\_bltzal    ? (pc\_plus\_4 + {{14{inst[15]}},inst[15:0],2'b0}) : 32'b0)

                        ;

## 4.3EX段

该段的输入输出总线定义如下方图片及表格所示：

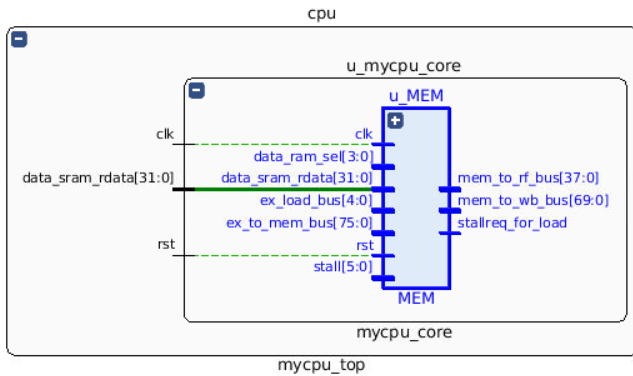


|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 接口名 | 宽度 | 输入/输出 | 作用 |
| 1 | clk | 1 | 输入 | 时钟信号 |
| 2 | rst | 1 | 输入 | 复位信号 |
| 3 | stall | 6 | 输入 | 控制暂停信号 |
| 4 | id\_to\_ex\_bus | 169 | 输入 | ID 段传给 EX 段的数据 |
| 5 | id\_load\_bus | 5 | 输入 | ID段传递读的数据 |
| 6 | id\_save\_bus | 3 | 输入 | ID段传递写的数据 |
| 7 | ex\_to\_mem\_bus | 80 | 输出 | EX 段传给 MEM 段的数据 |
| 8 | ex\_to\_rf\_bus | 38 | 输出 | EX 段传给regfile段的数据 |
| 9 | ex\_hi\_lo\_bus | 66 | 输出 | EX 段传给hilo段的数据 |
| 10 | stallreq\_for\_ex | 1 | 输出 | 对EX段的stall请求 |
| 11 | data\_sram\_en | 1 | 输出 | 内存数据的读写使能信号 |
| 12 | data\_sram\_wen | 4 | 输出 | 内存数据的写使能信号 |
| 13 | data\_sram\_addr | 32 | 输出 | 内存数据存放的地址 |
| 14 | data\_sram\_wdata | 32 | 输出 | 要写入内存的数据 |
| 15 | ex\_id | 38 | 输出 | EX 段传给 ID 段的数据 |
| 16 | data\_ram\_sel | 4 | 输出 | 内存数据的选择信号 |
| 17 | ex\_load\_bus | 5 | 输出 | EX 段读取的数据 |

EX段的任务也比较简单，主要负责计算从ID段传来的各类数据（存储于寄存器中，包括地址、数据等）。ALU已在库文件中提供，只需设置相关控制信号即可。

## **4.4** MEM段

该段的输入输出总线定义如下方图片及表格所示：



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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\_to\_mem\_bus | 80 | 输入 | EX 传给 MEM 段的数据 |
| 5 | data\_sram\_rdata | 32 | 输入 | 从内存中读出来要写入寄存器的值 |
| 6 | data\_ram\_sel | 4 | 输入 | 内存数据的选择信号 |
| 7 | ex\_load\_bus | 5 | 输入 | EX 段读取的数据 |
| 8 | stallreq\_for\_load | 1 | 输出 | 对EX段的stall请求 |
| 9 | mem\_to\_wb\_bus | 70 | 输出 | MEM 传给 WB 段的数据 |
| 10 | mem\_to\_rf\_bus | 38 | 输出 | MEM段传给regfile段的数据 |

主要任务是访存取数，核心逻辑如下：

    assign b\_data = data\_ram\_sel[3] ? data\_sram\_rdata[31:24] :

                    data\_ram\_sel[2] ? data\_sram\_rdata[23:16] :

                    data\_ram\_sel[1] ? data\_sram\_rdata[15: 8] :

                    data\_ram\_sel[0] ? data\_sram\_rdata[ 7: 0] : 8'b0;

    assign h\_data = data\_ram\_sel[2] ? data\_sram\_rdata[31:16] :

                    data\_ram\_sel[0] ? data\_sram\_rdata[15: 0] : 16'b0;

    assign w\_data = data\_sram\_rdata;

    assign mem\_result = inst\_lb     ? {{24{b\_data[7]}},b\_data} :

                        inst\_lbu    ? {{24{1'b0}},b\_data} :

                        inst\_lh     ? {{16{h\_data[15]}},h\_data} :

                        inst\_lhu    ? {{16{1'b0}},h\_data} :

                        inst\_lw     ? w\_data : 32'b0;

    assign rf\_wdata = sel\_rf\_res & data\_ram\_en ? mem\_result : ex\_result;

## **4.5** WB段

该段的输入输出总线定义如下方图片及表格所示：

图片包含 文本

描述已自动生成

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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 | wb\_to\_rf\_bus | 38 | 输出 | WB 传给 rf 的数据 |
| 6 | debug\_wb\_pc | 32 | 输出 | 用来 debug 的 pc 值 |
| 7 | debug\_wb\_rf\_wen | 4 | 输出 | 用来 debug 的写使能信号 |
| 8 | debug\_wb\_rf\_wnum | 5 | 输出 | 用来 debug 的写寄存器地址 |
| 9 | debug\_wb\_rf\_wdata | 32 | 输出 | 用来 debug 的写寄存器数据 |

该段任务是将数据写回内存，没有什么需要改动的地方。

核心逻辑：提取从MEM传入的相关控制信号，然后将写回信号发送给寄存器堆完成内存写入操作：

    assign {

        wb\_pc,

        rf\_we,

        rf\_waddr,

        rf\_wdata

    } = mem\_to\_wb\_bus\_r;

    assign wb\_to\_rf\_bus = {

        rf\_we,

        rf\_waddr,

        rf\_wdata

    };

## **4.6** STALL CTRL段

该段的输入输出总线定义如下方图片及表格所示：

图示

中度可信度描述已自动生成

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 接口名 | 宽度 | 输入/输出 | 作用 |
| 1 | clk | 1 | 输入 | 时钟信号 |
| 2 | stallreq\_for\_ex | 1 | 输入 | 执行阶段的指令是否请求流水线暂停 |
| 3 | stallreq\_for\_bru | 5 | 输入 | Load命令是否请求流水线暂停 |
| 4 | stall | 6 | 输出 | 暂停信号 |

Stall信号共6位，每一位分别代表流水线中某一段的暂停信号。stall[0]表示取指地址PC是否保持不变，为1表示保持不变；stall[1]表示流水线取指阶段是否暂停，为1表示暂停；stall[2]表示流水线译码阶段是否暂停，为1表示暂停；stall[3]表示流水线执行阶段是否暂停，为1表示暂停。stall[4]表示流水线访存阶段是否暂停，为1表示暂停；stall[5]表示流水线回写阶段是否暂停，为1表示暂停。

核心逻辑：

    always @ (\*) begin

        if (rst) begin

            stall = `StallBus'b0;

        end

        //stallreq\_for\_ex, stallreq\_for\_bru, stallreq\_for\_load

        else if (stallreq\_for\_ex) begin

            stall = `StallBus'b001111;

        end

        //

        else if (stallreq\_for\_bru) begin

            stall = `StallBus'b000111;

        end

        else if (stallreq\_for\_load) begin

            stall = `StallBus'b000011;

        end

        else begin

            stall = `StallBus'b0;

        end

    end

# 5 心得体会

**5.1张一航的心得体会**

在本次实验中，我主要负责了IF和ID模块的设计与实现。通过这一过程，我深入理解了五级流水线的工作机制以及各个阶段之间的相互依赖关系。特别是在实现数据前递和数据冒险处理时，我体验到了如何通过调整流水线的控制信号来避免冲突，确保指令能够顺利执行。尽管这部分实现较为复杂，但在不断调试和优化中，我逐渐掌握了流水线中各个控制逻辑的细节。

此外，这次实验让我更加熟悉了硬件描述语言（Verilog）的实际应用。通过编写和调试代码，我提高了自己的编程能力和逻辑思维能力。同时，实验过程中遇到的问题，如数据冒险导致的错误执行，促使我深入研究了相关的解决方法，如数据前递和流水线暂停机制，这不仅巩固了课堂所学知识，也拓展了我的实践经验。

在团队合作方面，我学到了如何高效地与组员沟通和协作。每个模块的设计和实现都需要紧密配合，及时分享进展和遇到的问题，确保整体设计的协调一致。这种协作方式不仅提高了工作效率，也增强了我的团队合作意识和解决问题的能力。

通过本次实验，我深刻体会到理论与实践的结合之重要。将课堂上学到的计算机体系结构知识应用到实际的CPU设计中，不仅加深了我对理论的理解，也让我认识到实际设计中可能遇到的各种挑战和解决方案。这为我未来深入学习计算机硬件设计和相关领域打下了坚实的基础。

## 5.2 王梓豪的心得体会

作为EX和MEM模块的负责人，我的主要任务是设计高效的ALU操作和地址计算模块，以及优化乘法和除法模块。在EX阶段，设计ALU并调整控制信号，以确保不同指令可以通过适当的操作数进行计算是一个非常有挑战性的任务。通过对各种指令的逐步处理，我学到了如何高效地设计运算模块，并掌握了如何将复杂的运算如乘法、除法整合到流水线中，以优化性能。

在MEM阶段，主要负责的访存操作让我更加深入地理解了计算机系统中内存与寄存器之间的数据交换机制。设计访存模块不仅需要考虑数据的正确读取和写入，还需要处理不同宽度的数据访问，如字节、半字和字的读写操作。这让我对内存访问的细节有了更全面的认识，并提升了我在数据选择和信号控制方面的能力。

整个设计过程中，我遇到了不少困难，例如在实现高效的地址计算时，如何确保地址的正确性和优化计算速度成为一个关键问题。通过查阅资料、与组员讨论和反复调试，我逐步解决了这些问题，优化了地址计算逻辑，提高了整体模块的性能。

此外，这次实验还提升了我的问题分析和解决能力。在设计和实现过程中，面对不断出现的错误和异常，我学会了如何系统性地进行排查和调试，逐步定位问题根源并加以解决。这不仅增强了我的技术能力，也培养了我在面对复杂问题时的耐心和细致。

团队合作方面，我与组员紧密配合，充分利用每个人的优势，共同推动项目的进展。通过协作，我们不仅提高了工作效率，也在相互学习中不断提升自己的专业技能。这种合作精神和团队意识将对我未来的学习和工作产生深远的影响。

5.3雷鹏的心得体会

在本次实验中，我负责了WB和CTRL模块的设计与实现。WB模块的任务相对简单，主要是将计算结果写回寄存器。不过，在进行这一部分的设计时，我理解了数据传递的重要性，尤其是在多个阶段之间如何高效传输数据。通过设计和实现WB模块，我熟悉了寄存器文件的操作流程，确保数据能够准确无误地写回寄存器，这对于整个流水线的正确运行至关重要。

CTRL模块的设计则更具挑战性，它涉及到流水线暂停的控制。设计一个高效且准确的控制模块，需要考虑来自EX段和BRU的多种stall请求，确保流水线在需要暂停时能够正确响应，并在条件满足后及时恢复。这部分工作让我深入理解了流水线控制的复杂性和重要性，也提升了我在逻辑设计和控制信号处理方面的能力。

在设计过程中，我遇到了如何有效管理多种stall请求的难题。通过分析需求和参考相关文献，我设计了一套合理的控制逻辑，确保不同类型的暂停请求能够被正确识别和处理。这不仅提高了流水线的稳定性，也优化了整体性能。

此外，实验过程中对调试工具的使用让我受益匪浅。通过Vivado的仿真功能，我能够直观地观察各个信号的变化，快速定位和解决问题。这种实践经验对于我未来的硬件设计工作具有重要的参考价值。

通过本次实验，我深刻体会到控制模块在流水线设计中的核心作用。设计一个高效的控制模块，不仅需要扎实的理论基础，还需要丰富的实践经验和敏锐的逻辑思维能力。这次实践让我在硬件设计的道路上迈出了坚实的一步，为我未来深入学习和研究计算机体系结构打下了坚实的基础。

# 6 结论

通过本次五级流水线CPU的设计与实现实验，我们团队成员深入理解了流水线技术在计算机体系结构中的应用，掌握了数据冒险和控制冒险的处理方法，熟悉了硬件描述语言（Verilog）的编写和调试过程。实验过程中，我们遇到了诸多挑战，但通过团队协作和不断优化，成功地完成了CPU的设计，并通过仿真验证了其功能的正确性。

此次实验不仅提升了我们的硬件设计能力和团队合作能力，也为我们今后深入学习计算机体系结构和数字电路设计打下了坚实的基础。未来，我们将继续探索更高级的流水线优化技术和硬件设计方法，提升自己的专业技能，迎接更复杂的设计挑战。

# 7 参考资料

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、（美）DavidA.Patterson、John L.Hennessy 著 《计算机组成与设计：硬件、软件接口（原书第 4 版）》

4、Yale N.Patt 著 《计算机系统概论（原书第 2 版）》

5、龙芯杯官方的参考文档

6、Verilog HDL官方文档及教程。