



**计算机组成原理 课 程 设 计**

团队实验报告

|  |  |
| --- | --- |
| 学 院 | 计算机学院 |
| 专 业 | 计算机科学与技术 |
| 指导老师 | 黄永刚 |
| 组 长 |  |
| 组 员 |  |
| 组长联系方式 |  |

二O二四年 九 月

目录

[第一章 项目简述 1](#_Toc176900302)

[第二章 组员分工 1](#_Toc176900303)

[第三章 设计目的 2](#_Toc176900304)

[第四章 设计环境 2](#_Toc176900305)

[第五章 设计原理及内容 2](#_Toc176900306)

[5.1 CPU整体架构 2](#_Toc176900307)

[5.1.1 数据通路 4](#_Toc176900308)

[5.1.2 控制逻辑 11](#_Toc176900309)

[5.2 流水线冒险问题以及解决方案 13](#_Toc176900310)

[5.2.1 冒险处理数据通路 13](#_Toc176900311)

[第六章 设计与实现 14](#_Toc176900312)

[第七章 测试 15](#_Toc176900313)

[7.1 指令测试 15](#_Toc176900314)

[Add、addi与sub的溢出 15](#_Toc176900315)

[Slt slti 15](#_Toc176900316)

[Sltu 16](#_Toc176900317)

[Sltiu 16](#_Toc176900318)

[And andi or ori xor xori 17](#_Toc176900319)

[Sll slli srl srli sra srai 18](#_Toc176900320)

[Beq bne 19](#_Toc176900321)

[Bltu 19](#_Toc176900322)

[Bgeu 20](#_Toc176900323)

[Jal和jalr 21](#_Toc176900324)

[Lb lh lw lbu lhu sb sh sw 21](#_Toc176900325)

[Lui 23](#_Toc176900326)

[Auipc 23](#_Toc176900327)

[7.2排序模块 24](#_Toc176900328)

[7.2.1 SortTop 模块 24](#_Toc176900329)

[7.2.2 DataMemory模块 24](#_Toc176900330)

[7.2.3 InstructionMemory模块 25](#_Toc176900331)

[7.2.4 ClockDivider 模块 25](#_Toc176900332)

[7.2.5 Num2Led 模块 25](#_Toc176900333)

[7.3下板测试 25](#_Toc176900334)

[第八章 问题及解决方法 28](#_Toc176900335)

[第九章 心得体会及总结 29](#_Toc176900336)

[第十章 参考文献有价值的资源推荐 30](#_Toc176900337)

# 项目简述

实现经典五级流水线CPU，支持41条riscv指令，包含冒险冲突处理，结合汇编与接口设计实验，合成一个能通过cpu控制并动态显示数组的冒泡排序过程的系统。

# 组员分工

小组分工见表2.1 小组分工。

|  |  |
| --- | --- |
| **组员** | **分工任务** |
| 组长 | * 负责CPU总体设计实现 * 外设部分的总体设计实现 * LCD模块设计实现 * CPU及外设联合设计编码 * 仿真及下板测试等 |
| 组员1 | * 负责CPU数据冒险处理 * CPU数据通路扩充 * 仿真及下板测试 * 结构流程图绘制 |
| 组员2 | * 负责CPU控制冒险处理 * 蜂鸣器音乐模块 * 仿真及下板测试 * PPT制作 |
| 组员3 | * 负责CPU指令扩充 * CPU逻辑功能扩充 * 功能仿真 * 文档报告撰写 |

表2.1 小组分工

# 设计目的

基于riscv指令集在上学期单周期cpu的基础上进行流水线cpu的设计，采用基础的五级流水线思路，提升cpu的性能效率，增加对cpu架构的理解，活用计算机组成原理的知识。

# 设计环境

|  |  |
| --- | --- |
| **操作系统** | Windows11 |
| **编程语言** | Verilog |
| **EDA工具** | Vivado2022.2 |
| **汇编语言** | riscv32i |
| **汇编程序编辑器** | VSCode |

# 设计原理及内容

## CPU整体架构

本流水线CPU覆盖41条指令（见表5.1-1），下面将从数据通路和控制逻辑阐述设计思路。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **序号** | **指令** | **opcode** | **Funct3** | **Funct7** | **功能** |
| 1 | ADD | 0110011 | 0x0 | 0x00 | rd=rs1+rs2 |
| 2 | MUL | 0110011 | 0x0 | 0x01 | rd=rs1\*rs2 |
| 3 | ADDI | 0010011 | 0x0 | / | rd=rs1+im |
| 4 | DIV | 0110011 | 0x0 | 0x02 | rd=rs1/rs2 |
| 5 | SUB | 0110011 | 0x0 | 0x20 | rd=rs1-rs2 |
| 6 | MOD | 0110011 | 0x0 | 0x03 | rd=rs1%rs2 |
| 7 | SLT | 0110011 | 0x2 | 0x00 | rd=(rs1<rs2)?1:0 |
| 8 | SLTI | 0010011 | 0x2 | / | rd=(rs1<imm)?1:0 |
| 9 | SLTU | 0110011 | 0x3 | 0x00 | rd=(rs1<rs2)?1:0 (无符号数) |
| 10 | SLTIU | 0010011 | 0x3 | / | rd=(rs1<imm)?1:0（无符号数） |
| 11 | AND | 0110011 | 0x7 | 0x00 | rd=rs1&r2 |
| 12 | ANDI | 0010011 | 0x7 | / | rd=rs1&imm |
| 13 | NOR | 000000 | 100111 |  | rd=!(rs1|rs2) |
| 14 | OR | 0110011 | 0x6 | 0x00 | rd=rs1|rs2 |
| 15 | ORI | 0010011 | 0x6 | / | rd=rs1|imm |
| 16 | XOR | 0110011 | 0x4 | 0x00 | rd=rs1 xor rd |
| 17 | XORI | 0010011 | 0x4 | / | rd=rs1 xor imm |
| 18 | SLL | 0110011 | 0x1 | 0x00 | rd=rs1<<r2 |
| 19 | SLLI | 0010011 | 0x1 |  | rd=rs1<<imm |
| 20 | SRL | 0110011 | 0x5 | 0x00 | rd=rs1>>r2 |
| 21 | SRLI | 000000 | 000110 |  | rd=rs>>imm |
| 22 | SRA | 0110011 | 0x5 | 0x20 | rd=rs1>>rs2（符号位保留） |
| 23 | SRAI | 0010011 | 0x5 |  | rd=rs1>>imm（符号位保留） |
| 24 | LB | 0000011 | 0x0 |  | rd = M[rs1+imm][0:7] |
| 25 | LH | 0000011 | 0x1 |  | rd = M[rs1+imm][0:15] |
| 26 | LW | 0000011 | 0x2 |  | rd = M[rs1+imm][0:31] |
| 27 | LBU | 0000011 | 0x4 |  | rd = M[rs1+imm][0:7] |
| 28 | LHU | 0000011 | 0x5 |  | rd = M[rs1+imm][0:15] |
| 29 | SB | 0100011 | 0x0 |  | M[rs1+imm][0:7] = rs2[0:7] |
| 30 | SH | 0100011 | 0x1 |  | M[rs1+imm][0:15] = rs2[0:15] |
| 31 | SW | 0100011 | 0x2 |  | M[rs1+imm][0:31] = rs2[0:31] |
| 32 | BEQ | 1100011 | 0x0 |  | PC=(rs1==rs2)?PC+im<<2:PC |
| 33 | BNE | 1100011 | 0x1 |  | PC=(rs1!=rs2)?PC+im<<2:PC |
| 34 | BLT | 1100011 | 0x4 | / | PC=(rs1<rs2)?PC+im<<2:PC |
| 35 | BGE | 1100011 | 0x5 | / | PC=(rs1>=rs2)?PC+im<<2:PC |
| 36 | BLTU | 1100011 | 0x6 | / | PC=(rs1<rs2)?PC+im<<2:PC |
| 37 | BGEU | 1100011 | 0x7 |  | PC=(rs1>=rs2)?PC+im<<2:PC |
| 38 | JAR | 1101111 | / | / | rd = PC+4; PC = rs1 + imm |
| 39 | JAL | 1101111 | / |  | rd = PC+4; PC += imm |
| 40 | LUI | 0110111 |  |  | rd = imm << 12 |
| 41 | AUIPC | 0010111 |  |  | rd = PC + (imm << 12) |

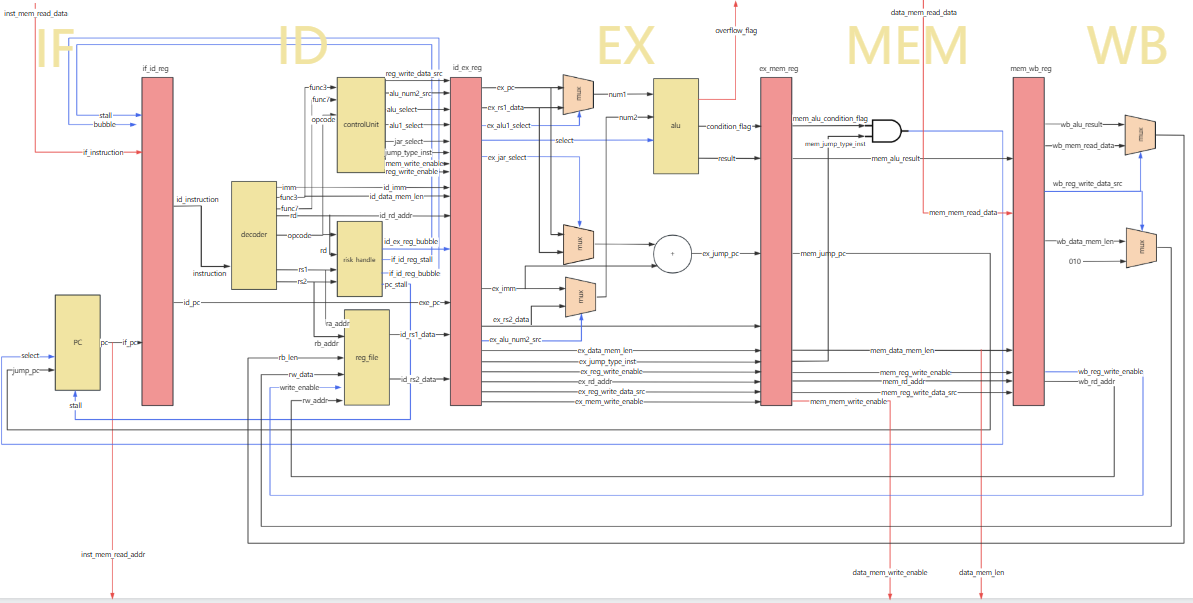
表5.1-1 34条指令

### 数据通路

流水线的设计有多种想法，不过当然按照最经典的划分想法将执行一条指令的过程分成五个阶段：取指、译码、执行、访存和写回。

在上学期单周期CPU的设计中我们也是将指令按照这五个步骤来进行的，只是在单周期中并不存在多个指令在不同阶段的并行执行。而要将单周期CPU向多周期CPU进行转变就是要能让多条指令的不同阶段能够做到并行执：首先就是要将整个执行逻辑拆分开来，打断各个子模块之间的“路径”；但是其之间的信号流通没有改变，因而要在每两个子模块间添加一组流水线寄存器，通过将上一条指令的前阶段得出的东西存入流水线寄存器从而使得下一条指令不用等待上一条指令完全完成就可以进入并且还不会干扰上一条指令的执行；进一步地，针对并行执行指令出现的数据冒险、控制冒险，设计添加数据、控制信号的转发路径，根据控制信号从各阶段数据寄存器中选择正确的数据参与运算。

总体架构图如下：



下面对每个阶段和每个阶段之间的流水线寄存器进行详细描述：

1. **取指阶段（IF\_stage）**

在取指阶段，程序计数器（PC）提供下一条指令的地址，该地址通过地址总线（addra）发送至指令存储器（InstMem）。指令存储器根据提供的地址输出对应的指令，该指令通过数据总线（douta）传输至IF\_ID流水线寄存器。同时，PC寄存器的内容也会被送入IF\_ID流水线寄存器，以便在下一周期使用。取指控制逻辑负责生成必要的控制信号，如使能信号（ena）和读使能信号（rdaddreac），以确保指令存储器正确地读取指令。此外，还包括复位信号（rstb）和时钟信号（clka），用于控制指令存储器的读写操作和同步。

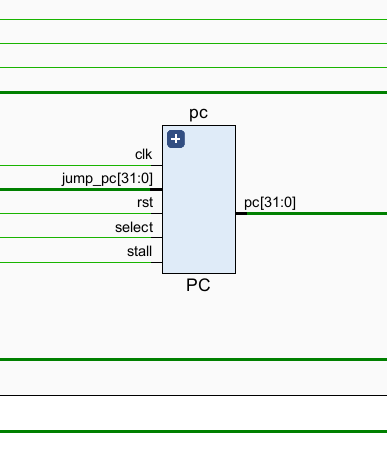
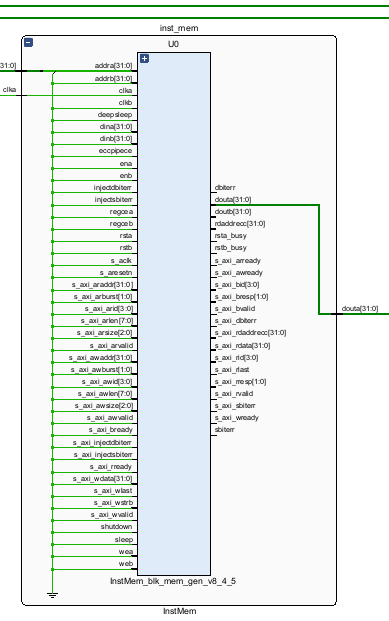
 

图5.1.1-1 取指阶段数据通路

1. **IF\_ID流水线寄存器**

在取指和译码阶段之间需要传递的信号是pc和读出的32位指令，因为在执行阶段计算下一个pc的值和跳转指令的目标地址时需要用到当前模块的pc所以pc需要逐级传递否则有可能被下一条指令更新丢失，保证pc和所在阶段执行的指令一致。IF\_ID流水线寄存器的作用是在CPU的流水线架构中暂存从指令存储器读取的指令以及当前的程序计数器（PC）值。IF\_ID流水线寄存器在取指阶段结束时捕获指令和PC值，为译码阶段做好准备，确保了流水线的连续性和指令的有序执行。

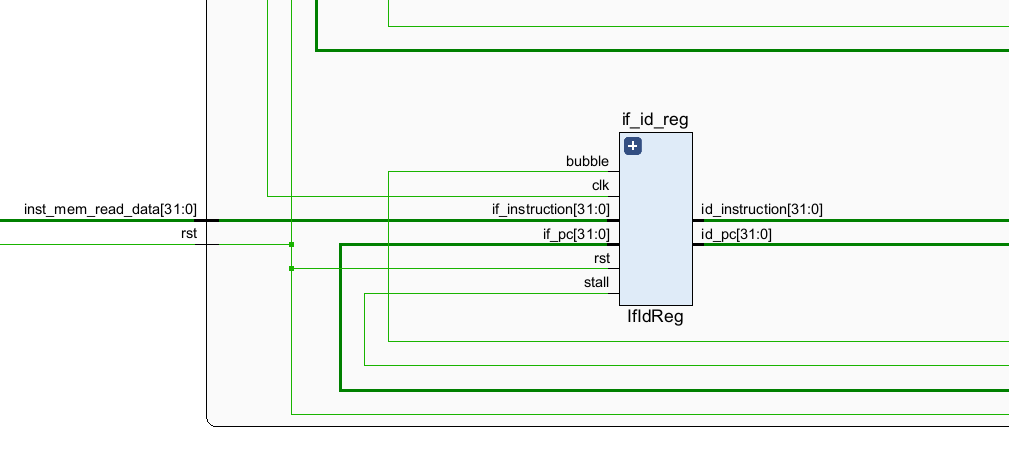


图5.1.1-2 IF\_ID流水线寄存器数据通路

1. **译码阶段（ID\_stage）**

译码阶段是CPU流水线中的关键环节，负责解析从IF\_ID流水线寄存器传来的指令，并从中提取出必要的信息，以便于后续执行阶段的操作。在这个阶段，指令的各个字段，包括操作码（op\_code）、寄存器地址（rd、rs1、rs2）、功能码（func3、func7）以及立即数（imm），被解码器（Decoder）解析。解码器接收来自IF\_ID寄存器的完整指令（instruction），并根据指令的操作码和功能码生成相应的控制信号。这些控制信号将指导执行阶段的算术逻辑单元（ALU）进行正确的操作。

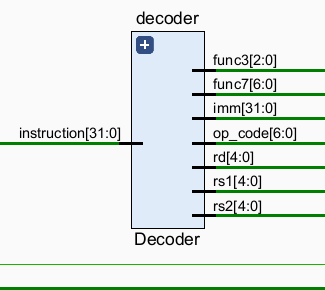


图5.1.1-3 译码阶段数据通路

1. **寄存器堆（regfile）**

在每个时钟周期，如果write\_enable信号被激活，寄存器堆会将rw\_data中的数据写入rw\_addr指定的寄存器。同时，寄存器堆会根据ra\_addr和rb\_addr提供的地址，从内部寄存器中读取数据，并通过ra\_data和rb\_data输出。

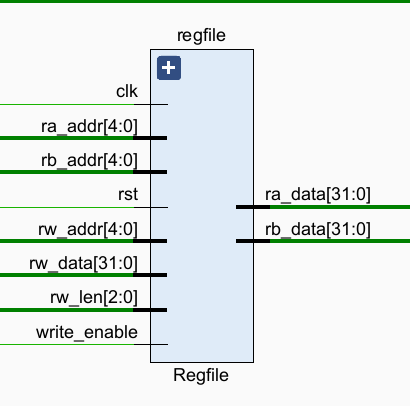


图5.1.1-4 寄存器堆数据通路

1. **控制模块（controlunit）**

控制单元（Control Unit）负责解码指令并生成控制信号以指导其他硬件组件的操作。它根据指令的操作码（op\_code）和功能码（func3, func7）来确定需要执行的操作类型。控制单元生成的信号包括ALU选择信号（alu\_select），它告诉算术逻辑单元（ALU）执行特定的算术或逻辑操作；ALU第二个操作数来源选择（alu\_num2\_src），它决定ALU的第二个操作数是来自寄存器还是立即数；以及寄存器写入使能（reg\_write\_enable），内存写入使能（mem\_write\_enable），寄存器写入数据来源选择（reg\_write\_data\_src），跳转类型指示（jump\_type\_inst），以及JALR指令选择（jar\_select）。

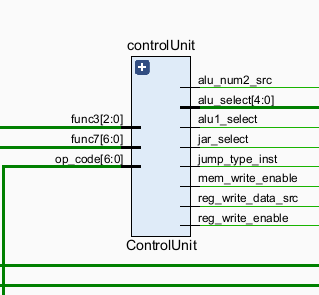


图5.1.1-5 控制模块数据通路

1. **ID\_EX流水线寄存器**

ID\_EX流水线寄存器是位于译码阶段（ID）和执行阶段（EX）之间的关键组件，负责暂存译码阶段产生的控制信号和数据，以便在执行阶段使用。它接收来自译码阶段的多个输入信号，包括立即数（id\_imm）、寄存器数据（id\_rs1\_data、id\_rs2\_data）、目标寄存器地址（id\_rd\_addr）、ALU选择信号（id\_alu\_select）、寄存器写入使能（id\_reg\_write\_enable）、内存写入使能（id\_mem\_write\_enable）、ALU第二个操作数来源（id\_alu\_num2\_src）、写回数据来源选择（id\_reg\_write\_data\_src）、跳转类型指示（id\_jump\_type\_inst）、程序计数器值（id\_pc）、JALR指令选择（id\_jar\_select）、ALU操作选择（id\_alu1\_select）以及数据存储器长度（id\_data\_mem\_len）。

当复位信号（rst）为低时，所有输出寄存器被清零。如果接收到气泡信号（bubble），则在下一个周期插入气泡，用于处理流水线中的冒险和冲突。在正常操作中，ID\_EX寄存器将译码阶段的输出信号传递到执行阶段的对应输入信号。

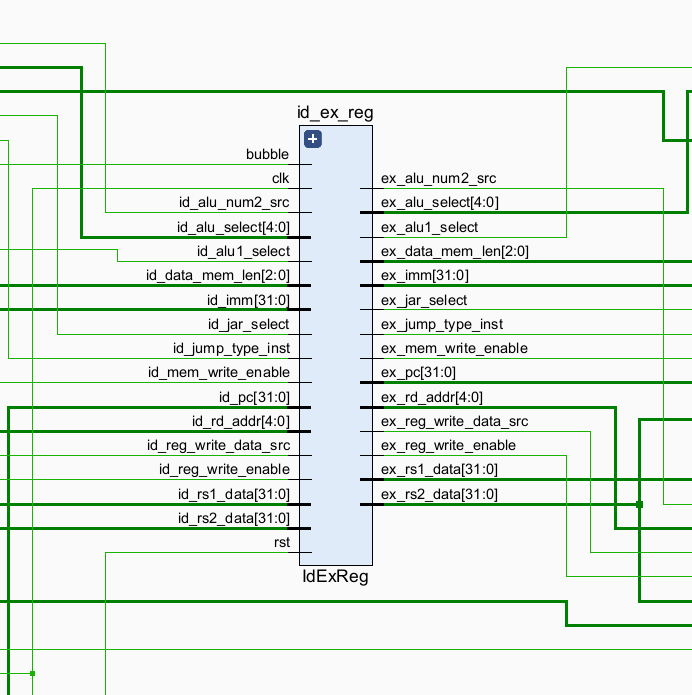


图5.1.1-6 ID\_EX流水线寄存器数据通路

1. **执行阶段（EX\_stage）**

该阶段要做的事情就是alu的算术运算以及pc新的值的计算（包含得到跳转的地址值），可以说alu的结果一般是算术运算指令的结果、新的pc地址等，而得到什么结果取决于不同的数据来源也就是上个阶段传递过来的信号和控制器传递的控制信号。

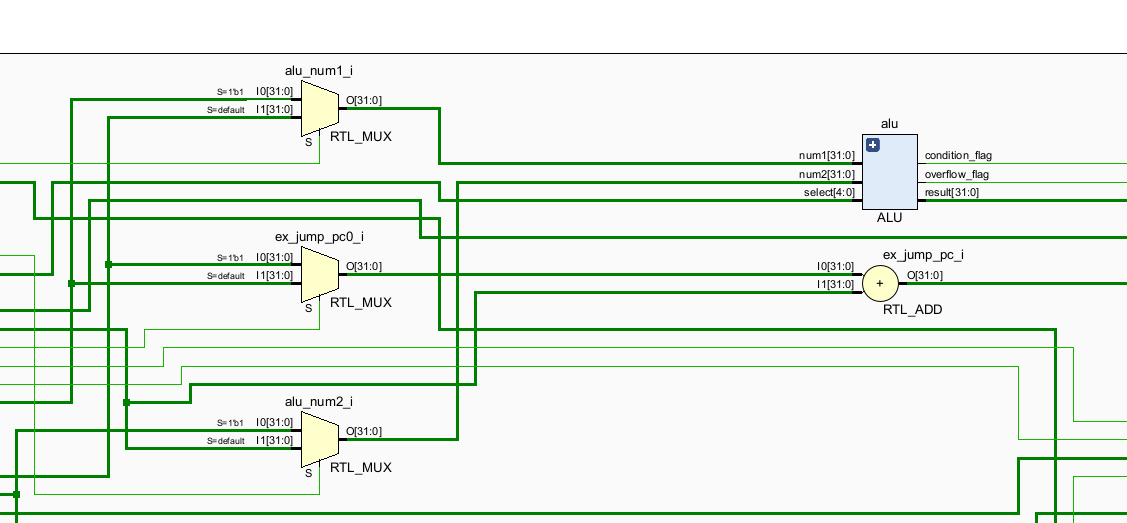


图5.1.1-7 执行阶段数据通路

1. **EX\_MEM流水线寄存器**

暂存执行阶段产生的结果和控制信号，包括ALU的结果（ex\_alu\_result）、条件标志（ex\_alu\_condition\_flag）、跳转目标PC（ex\_jump\_pc）、跳转类型指令（ex\_jump\_type\_inst）、内存写入使能（ex\_mem\_write\_enable）、目标寄存器地址（ex\_rd\_addr）、寄存器写入数据来源（ex\_reg\_write\_data\_src）、寄存器写入使能（ex\_reg\_write\_enable）、以及需要访问的寄存器数据（ex\_rs2\_data），其中alu\_result会在访存阶段作为访存对应的地址使用，在写回阶段作为运算结果写入写回寄存器，jump\_pc和jump\_type\_inst则是在写回阶段作为pc新值的待选写回寄存器，rs2\_data则是在访存阶段作为写入存储器的数据使用。

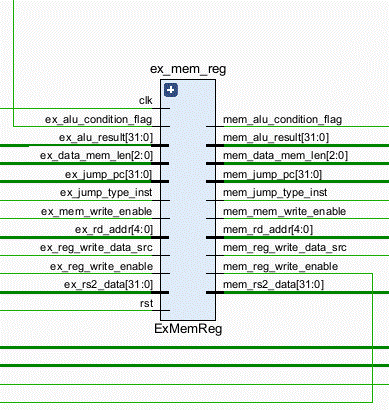


图5.1.1-8 EX\_MEM流水线寄存器数据通路

1. **访存阶段（MEM\_stage）**

根据执行阶段产生的控制信号和地址信息，处理器会进行数据的读取或写入操作。具体来说，如果当前执行的是加载指令，访存阶段会从数据存储器中读取数据，并将其暂存，以便在写回阶段（WB\_stage）写入指定的寄存器。相反，如果执行的是存储指令，访存阶段会将数据从寄存器读取出来，并将其写入到数据存储器的指定位置。

1. **MEM\_WB流水线寄存器**

在访存阶段，数据存储器根据执行的操作（如加载或存储）进行读写，而MEM\_WB流水线寄存器则保存了这些操作的结果。具体来说，它保存了ALU的结果（alu\_result）、数据存储器的长度（data\_mem\_len）、从数据存储器读取的数据（read\_data）、目标寄存器地址（rd\_addr）、寄存器写入数据来源选择（reg\_write\_data\_src）以及寄存器写入使能信号（reg\_write\_enable）。

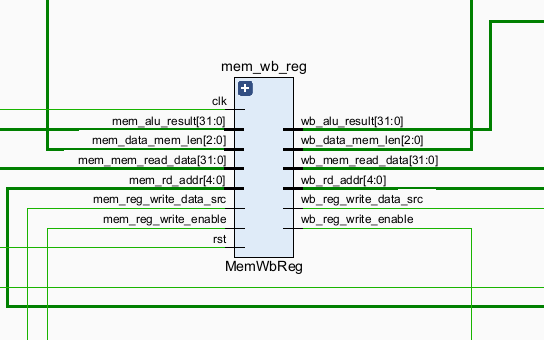


图5.1.1-9 MEM\_WB流水线寄存器数据通路

1. **写回阶段（WB\_stage）**

检查写回使能信号（reg\_write\_enable），以确定是否需要更新寄存器。如果使能信号有效，数据将根据指令中指定的目的寄存器地址（rd\_addr）写入相应的寄存器。写回的数据可以是算术逻辑单元（ALU）的结果或内存访问的结果，这一选择由数据来源选择信号（reg\_write\_data\_src）控制。写回操作完成后，相关寄存器将包含最新的指令执行结果，供后续指令使用或作为程序状态的一部分。

### 控制逻辑

流水线cpu的绝大部分控制信号和单周期cpu类似，采用组合逻辑硬联产生，控制器都是一样的，需要注意哪些控制信号需要随着流水线寄存器进行保存。

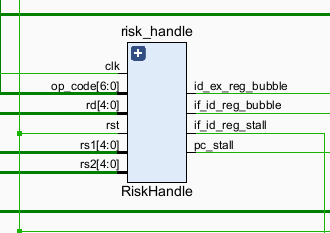
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **信号** | **位宽** | **输入/输出** | **描述** |
| 1 | inst\_mem\_read\_data | 32 | Input | 指令 |
| 2 | inst\_mem\_read\_addr | 32 | Output | 指令存储器的读取地址 |
| 3 | data\_mem\_write\_enable | 1 | Output | 写数据使能 |
| 4 | data\_mem\_addr | 32 | Output | 提供数据存储器的读写地址 |
| 5 | data\_mem\_write\_data | 32 | Output | 写入的数据 |
| 6 | data\_mem\_len | 3 | Output | 数据长度 |
| 7 | data\_mem\_read\_data | 32 | Input | 读取的数据 |
| 8 | reg\_write\_enable | 1 | Output | 寄存器写入使能 |
| 9 | reg\_ra\_addr | 5 |  | 第一个源操作数寄存器的地址 |
| 10 | reg\_rb\_addr | 5 |  | 第二个源操作数寄存器的地址 |
| 11 | reg\_rw\_addr | 5 |  | 目标寄存器地址 |
| 12 | reg\_rw\_data | 32 |  | 要写入到寄存器堆的数据 |
| 13 | reg\_ra\_data | 32 |  | 第一个源寄存器的数据 |
| 14 | reg\_rb\_data | 32 |  | 第二个源寄存器的数据 |
| 15 | reg\_ data\_mem\_len | 3 |  | 控制数据存储器访问长度 |
| 16 | Opcode | 7 |  | 操作码 |
| 17 | rd\_addr | 5 |  | 目的寄存器地址 |
| 18 | rs1\_addr | 5 |  | 第一个源操作数寄存器的地址 |
| 19 | rs2\_addr | 5 |  | 第二个源操作数寄存器的地址 |
| 20 | pc\_stall | 1 |  | 控制暂停PC更新 |
| 21 | if\_id\_reg\_stall | 1 |  | 控制暂停IF/ID数据传输 |
| 22 | if\_id\_reg\_bubble | 1 |  | 控制IF/ID气泡插入 |
| 23 | id\_ex\_reg\_bubble | 1 |  | 控制ID/EX气泡插入 |

需要注意的是控制信号的保存规律。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **序号** | **信号** | **IF** | **ID** | **EX** | **MEM** | **WB** |
| 1 | inst\_mem\_read\_data | 1 |  |  |  |  |
| 2 | inst\_mem\_read\_addr | 1 |  |  |  |  |
| 3 | data\_mem\_write\_enable |  |  |  | 1 |  |
| 4 | data\_mem\_addr |  |  |  | 1 |  |
| 5 | data\_mem\_write\_data |  |  |  | 1 |  |
| 6 | data\_mem\_len |  |  |  | 1 |  |
| 7 | data\_mem\_read\_data |  |  |  | 1 |  |
| 8 | reg\_write\_enable |  | 1 |  |  | 1 |
| 9 | reg\_ra\_addr |  | 1 |  |  |  |
| 10 | reg\_rb\_addr |  | 1 |  |  |  |
| 11 | reg\_rw\_addr |  |  |  |  | 1 |
| 12 | reg\_rw\_data |  |  |  |  | 1 |
| 13 | reg\_ra\_data |  | 1 |  |  |  |
| 14 | reg\_rb\_data |  | 1 |  |  |  |
| 15 | reg\_ data\_mem\_len |  |  |  |  | 1 |
| 16 | Opcode |  | 1 |  |  |  |
| 17 | rd\_addr |  | 1 |  |  |  |
| 18 | rs1\_addr |  | 1 |  |  |  |
| 19 | rs2\_addr |  | 1 |  |  |  |
| 20 | pc\_stall | 1 |  |  |  |  |
| 21 | if\_id\_reg\_stall |  | 1 |  |  |  |
| 22 | if\_id\_reg\_bubble |  | 1 |  |  |  |
| 23 | id\_ex\_reg\_bubble |  |  | 1 |  |  |

## 流水线冒险问题以及解决方案

单独设计了冒险处理模块（RiskHandle），它通过分析当前指令和前几个周期指令的寄存器依赖关系，来确定是否存在数据冒险。该模块接收当前指令的操作码（op\_code）、目的寄存器（rd）、源寄存器1（rs1）和源寄存器2（rs2），并根据这些信息生成控制信号，如程序计数器暂停（pc\_stall）、IF\_ID寄存器暂停（if\_id\_reg\_stall）、IF\_ID寄存器插入气泡（if\_id\_reg\_bubble）和ID\_EX寄存器插入气泡（id\_ex\_reg\_bubble）。



### 5.2.1 冒险处理数据通路

1. **数据冒险**

数据冒险出现的主要原因是：由于流水线cpu并发指令多条指令，可能导致流水线上后一条指令需要使用前一条指令的运算结果，共4种类型。

* 1. 在一个周期开始，EX 阶段要使用上一条处在 EX 阶段指令的执行结果，此时需要将 EX/MEM 寄存器的数据前递。
  2. 在一个周期开始，EX 阶段要使用上一条处在 MEM 阶段指令的执行结果，此时我们将 MEM/WB 寄存器的数据前递。
  3. 在一个周期开始，EX 阶段要使用上一条处在 WB 阶段指令的执行结果，此时使用寄存器堆前递机制（即在clk信号下降沿写入寄存器）。
  4. 在发生load-use型冒险的时候，如果是load后跟着store指令，并且load指令的rd与store指令的rs1 不同而与rs2相同，此时为load-use的特殊情形：不需要停顿流水线，而将MEM/WB 寄存器的数据前递到MEM阶段。

如果检测到数据冒险，根据冒险的类型和位置，模块会决定是否需要暂停流水线的特定阶段或插入气泡来解决冒险。

1. **控制冒险**

控制冒险指流水线CPU在处理条件分支指令时，由于跳转条件是否成立需要在执行阶段才能确定，而流水线会不断流入后续指令，导致出现流入流水线的后续指令并非程序所需要的指令的情况。出现控制冒险时需要对流水线进行停顿或者冲刷，来处理流入流水线的多余指令。

在本CPU中，当检测到操作码属于分支（B\_TYPE\_OP\_CODE）、跳转（JAL\_TYPE\_OP\_CODE）或跳转寄存器（JALR\_TYPE\_OP\_CODE）指令时，模块通过设置pc\_stall信号来保持程序计数器（PC）在当前指令的下一条指令位置，同时使用if\_id\_reg\_bubble信号在IF\_ID寄存器中插入气泡，取消该指令之后的所有指令执行，直到分支结果确定。此外，jump\_stall\_counter计数器用于跟踪因控制冒险而需要暂停的周期数，确保在分支目标地址确定之前，流水线不会错误地继续执行无关的指令。

# 设计与实现

具体设计的思路已经在数据通路阶段进行了详细说明。设计并实现了以下模块：

1. PipelineCPU：顶层模块，负责连接cpu和存储器进行数据交换，包含模块：regfile、risk\_handle
2. IF:取指阶段，根据pc将对应的指令从指令存储器中取出，包含模块：pc
3. ID:译码阶段，将取出的指令进行译码，得到控制信号和指令中的数据，指引指令的执行方式。包含模块：if\_id\_reg、decoder、controlunit
4. EX:执行阶段，进行alu计算，跳转指令执行pc计算，计算指令执行逻辑、移位等运算，根据计算结果判定分支，包含模块:id\_ex\_reg、alu
5. MEM:访存阶段，与外部存储器进行数据交换的阶段，包含模块：ex\_mem\_reg
6. WB:写回阶段，选择要写回的数据并向寄存器堆写入数据，包含模块：mem\_wb\_reg

# 测试

## 7.1 指令测试

### Add、addi与sub的溢出

两个正数相加，得到负数，溢出

图形用户界面, 应用程序, Word

描述已自动生成图形用户界面

描述已自动生成负数减正数，得到正数，溢出

### Slt slti

表格

中度可信度描述已自动生成

图形用户界面

描述已自动生成

### Sltu

Rs1=8, rs2=-1

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成按无符号数比较，rd=1

图形用户界面

描述已自动生成

### Sltiu

1与-1比较

图形用户界面, 应用程序, 表格, Word

中度可信度描述已自动生成由于无符号数，因此rd=1

电脑萤幕的截图

描述已自动生成

### And andi or ori xor xori

表格

描述已自动生成

电视游戏的萤幕截图

描述已自动生成

### Sll slli srl srli sra srai

sll

图形用户界面, 应用程序

描述已自动生成

图形用户界面, 应用程序

描述已自动生成

Slli

表格

中度可信度描述已自动生成

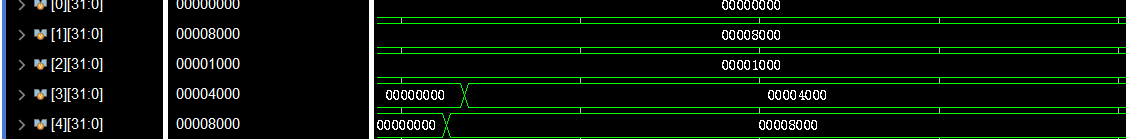
图形用户界面

描述已自动生成

Srl、srli

图形用户界面, 应用程序, 表格

描述已自动生成



Sra、srai

图形用户界面, 应用程序, 表格

描述已自动生成

图形用户界面

描述已自动生成

### Beq bne

表格

描述已自动生成

图形用户界面

描述已自动生成

### Bltu

发生跳转，赋值语句未运行图形用户界面, 应用程序, 表格, Word

描述已自动生成

图形用户界面

描述已自动生成

### Bgeu

发生跳转，赋值语句未运行

图形用户界面

描述已自动生成图形用户界面, 应用程序

描述已自动生成

### Jal和jalr

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

图形用户界面

描述已自动生成X1, x2, pc的值均正确变化

### Lb lh lw lbu lhu sb sh sw

图形用户界面

中度可信度描述已自动生成

日程表

描述已自动生成

图形用户界面

中度可信度描述已自动生成

### Lui

图形用户界面, 文本, 应用程序, 表格

描述已自动生成

电脑萤幕截图

描述已自动生成

### Auipc

图形用户界面, 应用程序

中度可信度描述已自动生成

图形用户界面, 表格

描述已自动生成

## 7.2排序模块

### 7.2.1 SortTop 模块

SortTop模块是顶层模块，它整合了整个排序系统的功能，包括排序算法的执行和结果的显示。它接收一个时钟信号（clk）和复位信号（rst），控制整个系统的启动和重置。

该模块通过PipelineCPU实例来执行排序算法，使用DataMemory来存储初始数据和排序过程中的中间结果，以及InstructionMemory来存储排序算法的指令。此外，它还包含时钟分频器模块ClockDivider，用于调整时钟频率以便于观察排序过程和控制数码管的显示。SortTop模块还负责将排序结果通过数码管显示出来，使用Num2Led模块将数字转换为数码管的显示编码。

### 7.2.2 DataMemory模块

该模块使用一个简单的RAM模型来保存排序算法需要操作的数据元素。该模块在初始化时加载了一系列降序排列的数值到RAM中，这些数值作为冒泡排序算法的输入数据。写操作由write\_enable信号控制，当该信号有效时，表示CPU想要向指定的地址addr写入数据write\_data。写数据会根据提供的地址被放入RAM的相应位置。读操作则是通过addr信号来指定想要读取数据的地址，然后通过read\_data信号返回存储在该地址的数据。

### 7.2.3 InstructionMemory模块

在系统初始化时，通过一系列的赋值操作，将冒泡排序所需的指令编码填充到InstructionMemory\_中。这些指令被编码在模块内部，并在复位时加载。当PipelineCPU请求指令时，InstructionMemory\_会根据提供的地址返回相应的指令。

### 7.2.4 ClockDivider 模块

ClockDivider模块是一个时钟分频器，它接收一个原始的快速时钟信号（clk），并生成一个频率较慢的时钟信号（clk\_slow）。这是通过计数原始时钟周期并在达到特定阈值时翻转输出时钟信号来实现的。该模块用于调整PipelineCPU和数码管的时钟频率，使得排序过程和数码管的显示速度减慢，便于观察和调试。

### 7.2.5 Num2Led 模块

负责将数字转换为数码管的显示编码。它接收一个4位的数字输入，并根据这个数字生成一个7段数码管的显示模式。每个数字对应一组特定的段选信号，这些信号控制数码管上的LED灯的亮灭，从而显示出相应的数字。

## 7.3下板测试

以一个按从小到大顺序排列的数组为例，最后实现将其按从大到小的顺序。

汇编代码

*# x1寄存器存放常数 ((8-1)\*4)*

    addi %t1, %zero, 28

*# x2 寄存器存放数组起始内存地址，这里为 0*

    addi x2, x0, 0

*# x3寄存器存放外层循环变量 i = 28*

    addi x3, x1, 0

loop:

*# x4 寄存器存放内层循坏变量 j = 0*

    addi x4, x0, 0

inner\_loop:

*# x5 寄存器存放当前访问内存数组元素的地址*

    add x5, x2, x4

*# x6 存放当前访问内存数组元素值*

    lw x6, 0(x5)

*# x7 存放下一个内存数组元素值*

    lw x7, 4(x5)

    blt x6, x7, inner\_loop\_end

*# 交换两个相邻数组元素*

    sw x7, 0(x5)

    sw x6, 4(x5)

inner\_loop\_end: *# 内循环结束处理，j+4*

    addi x4, x4, 4

    blt x4, x3, inner\_loop

loop\_end: *# 外循环结束处理，i-4*

    addi x3, x3, -4

    bge x3, x0, loop

*# 让程序停止在这里*

finish:

    bge x0, x0, finish

机器码

01c00093

00000113

00008193

00000213

004102b3

0002a303

0042a383

00734663

0072a023

0062a223

00420213

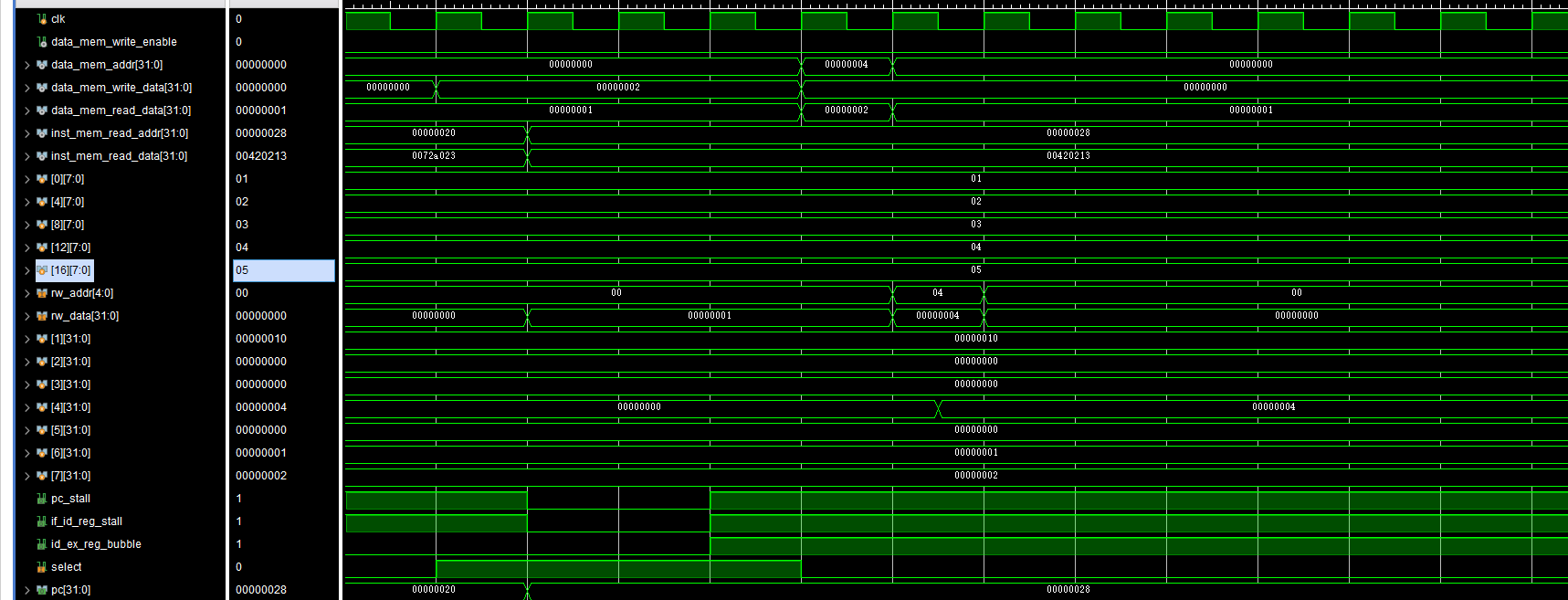
fe3242e3

ffc18193

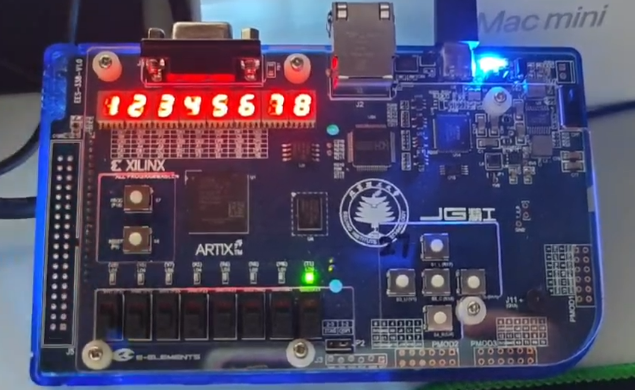
fc01dce3

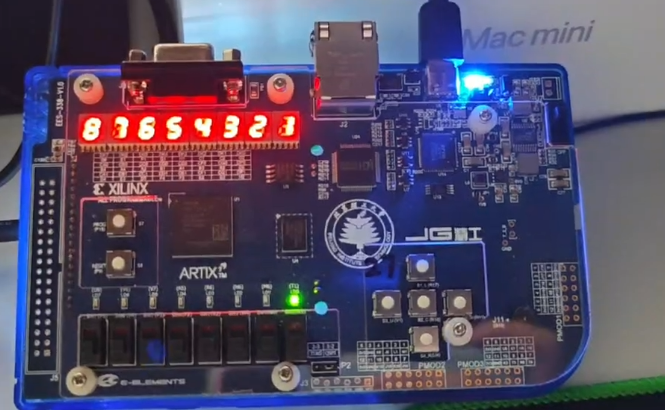
00005063

仿真结果



下板测试结果





# 问题及解决方法

在设计流水线 CPU 的过程中，我们主要遇到了数据相关和控制相关不能正确执行的问题。最初的设计中，CPU 的流水线存在很多bug，通过查阅参考书籍和反复梳理设计逻辑，最终找到了合适的解决方案。

**数据相关问题的处理**

数据相关问题的核心是当指令之间存在数据依赖时，后续指令需要等待前面的指令完成计算，才能继续执行。为了解决这个问题，我通过在流水线中插入气泡来暂停部分指令的执行。具体方法是通过一个模块同时控制 PC、IF/ID、ID/EX 阶段的信号。当检测到数据相关时，将 PC 和 IF/ID 阶段的 stall 信号置为 1，ID/EX 阶段的 bubble 信号置为 1，从而在 EX 阶段插入一个气泡，等待相关的数据准备好。

**控制相关问题的处理**

控制相关问题主要发生在条件跳转指令中，这类指令会影响流水线中后续指令的执行顺序。为了解决这个问题，我们采用了向译码阶段插入气泡的方式。当控制相关问题出现时，将 PC 的 stall 信号置为 1，IF/ID 阶段的 bubble 信号置为 1，以保证正确的控制流，避免错误的指令进入后续流水线阶段。

通过这样的设计，成功解决了流水线中的数据相关和控制相关问题，保证了流水线的正确执行。

# 心得体会及总结

本次流水线 CPU 的设计与实现本小组完全从零开始并独立完成实验项目，经过我们团队的共同努力，最终成功实现了流水线CPU并运行了冒泡排序测试程序。通过这次实验，我们不仅实现了硬件设计的目标，也收获了诸多宝贵的经验和知识。

在整个项目开发过程中，团队合作发挥了至关重要的作用。从初期设计到中期调试，再到最终的实现，每个环节都需要团队成员的紧密配合与协调。小组成员在遇到问题时也会相互讨论与帮助。通过这种合作开发的方式，团队成员学会了如何有效沟通，如何在遇到复杂问题时集思广益，最终解决问题。项目的推进让我们深刻体会到了团队协作在实际工程开发中的重要性。

在本次实验中，我们将之前在计算机组成原理课程中学到的知识真正应用到了实践中，特别是关于 CPU 数据通路、流水线控制和指令周期等内容。通过将这些理论知识与实际硬件设计相结合，我们加深了对这些概念的理解。实验过程中遇到的时序问题、数据相关问题以及控制相关问题，都是我们在课堂上学习的理论问题，但通过实际的硬件设计和调试，我们真正掌握了如何解决这些问题。

在设计流水线 CPU 的过程中，时序和数据流的控制是最具挑战的部分。特别是数据相关和控制相关问题对流水线性能的影响尤为显著。我们通过为数据相关插入气泡，暂停部分流水线级，以及为控制相关问题插入气泡，调整控制流等方式，成功解决了这些复杂问题。这样的过程不仅让我们对流水线结构有了更深入的理解，也加深了对数据通路中每一阶段的运作机制的认识。

在解决流水线时序问题时，我们学会了用系统化、条理化的思维方式来整理每个周期中各个指令的执行情况。通过使用表格、列表等工具对每个时钟周期的流水线执行情况进行梳理，我们能够清楚地看到每条指令在每个阶段的进展情况，找出问题的根源并进行修正。这种方法大大提高了我们调试代码的效率，也让我们在处理复杂时序问题时更为得心应手。

实验的最终成果非常令人满意，CPU 成功运行了冒泡排序程序，流水线设计验证了我们的理论假设和设计思路。在实验过程中，我们也意识到，除了技术问题外，实验项目的推进还需要细致的规划和合理的时间管理。遇到复杂问题时，要有足够的耐心，并采取多种方法尝试解决。

通过本次流水线 CPU 的设计与实现，我们不仅提升了实际动手能力，也巩固了计算机硬件系统设计的核心知识。这次实验无疑是一次极具价值的实践经历。

# 参考文献有价值的资源推荐

[CS:APP3e, Bryant and O'Hallaron (cmu.edu)](http://csapp.cs.cmu.edu/)

<https://zanpu.spencerwoo.com/>

[GitHub - isrc-cas/riscv-isa-manual-cn](https://github.com/isrc-cas/riscv-isa-manual-cn)