|  |
| --- |
|  |
| 中期报告 |
| C++实现RISC-V模拟器 |
|  |
| **组长：薛皓天 210340170**  **组员：薛傲翔 210340169**  **杨馥银 210340172** |
|  |

**目录**

[一、任务分配 2](#_Toc178111649)

[1.1薛皓天（组长） 3](#_Toc178111650)

[1.2 薛傲翔（组员） 3](#_Toc178111651)

[1.3杨馥银（组员） 3](#_Toc178111652)

[二、模型机简要说明 3](#_Toc178111653)

[2.1结构特点 3](#_Toc178111654)

[2.2设计目的 3](#_Toc178111655)

[2.3技术采用 3](#_Toc178111656)

[2.4部件示意图 4](#_Toc178111657)

[三、指令集设计 4](#_Toc178111658)

|  |
| --- |
|  |

# 一、任务分配

## 1.1薛皓天（组长）

总体架构规划；代码编写；最终报告攥写；文档书写。

## 1.2 薛傲翔（组员）

中期报告攥写；代码测试和纠错。

## 1.3杨馥银（组员）

资料收集。

# 二、模型机简要说明

## 2.1结构特点

由于选题为C++实现的RISC-V模拟器，因此基本结构需要包括寄存器堆、内存和指令集模块。利用C++的面向对象特性，将模拟器的特定的部分封装成类。例如，指令可以定义为类的实例，每个类对应一种指令类型，包含执行该指令所需的逻辑。利用C++的强大的库函数，其中寄存器堆可以采用哈希表（unordered\_map）的数据结构所表示，用于映射寄存器编号到其值，以支持快速访问和可能的寄存器重命名技术，同样内存也可以使用如上的数据结构进行表示。

## 2.2设计目的

1、通过亲手编写模拟器，可以更深入地理解RISC-V指令集架构的内部工作机制，包括指令的执行流程、寄存器的使用、内存访问等关键概念。这将有助于我们在未来的学习和工作中更好地应用RISC-V架构。

2、编写RISC-V模拟器是一个将计算机体系结构理论知识付诸实践的好机会。我们将能够应用所学到的关于处理器设计、指令集、流水线、缓存、存储层次结构等概念，通过编程实现这些概念在模拟器中的具体表现。

3、 在编写模拟器的过程中，我们实现了不同的RISC-V扩展指令集，以探索这些扩展如何影响处理器的性能和功能。这将有助于你们更深入地理解RISC-V架构的灵活性和可扩展性，并为未来的研究和开发提供经验。

## 2.3技术采用

采用C++进行编写，RISC-V模拟器的开发充分利用了C++作为一种高效、灵活且功能强大的编程语言的优势。C++不仅提供了面向对象的编程范式，使得模拟器的各个组成部分（如寄存器堆、内存和指令集模块）可以自然地封装成类，还通过其丰富的标准库和强大的类型系统支持复杂的内存管理、数值计算以及位操作等需求。

在编写过程中，我们小组可以利用C++的类和对象来构建模拟器的核心数据结构，如通过std::unordered\_map实现高效的寄存器查找和内存访问。同时，C++的模板特性允许我们小组编写通用的函数和数据结构，以支持不同大小和类型的数据操作，从而提高代码的可重用性和可维护性。

## 2.4部件示意图

如图1所示为本模拟器的部件示意图，该示意图详细展示了计算机系统的基本组成，包括输入设备、CPU（中央处理器）、存储器、输出设备以及控制单元。这些关键部件通过箭头紧密相连，形象地描绘了数据在它们之间的流动过程，共同构成了计算机模拟器的核心架构。

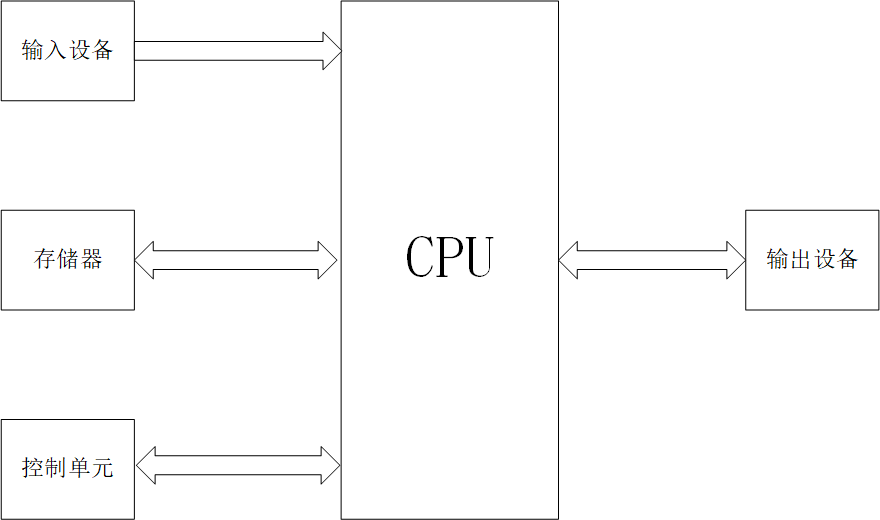


图1 模拟器部件图

# 三、指令集设计

如表1和图2所示为一套指令集，它列出了各种指令的类型及其对应的功能。这些指令是计算机程序执行的基础，它们通过不同的方式控制着计算机硬件的操作流程。

表1各种指令的功能

|  |  |  |
| --- | --- | --- |
| 序号 | 指令类型 | 功能 |
| 1 | R型指令 | 用于寄存器和寄存器操作 |
| 2 | I型指令 | 用于短立即数和内存载入指令load操作 |
| 3 | S型指令 | 用于内存存储store操作 |
| 4 | B（SB）指令 | 用于有条件跳转操作 |
| 5 | U型指令 | 用于长立即数操作 |
| 6 | J（UJ）型指令 | 用于无条件跳转操作 |

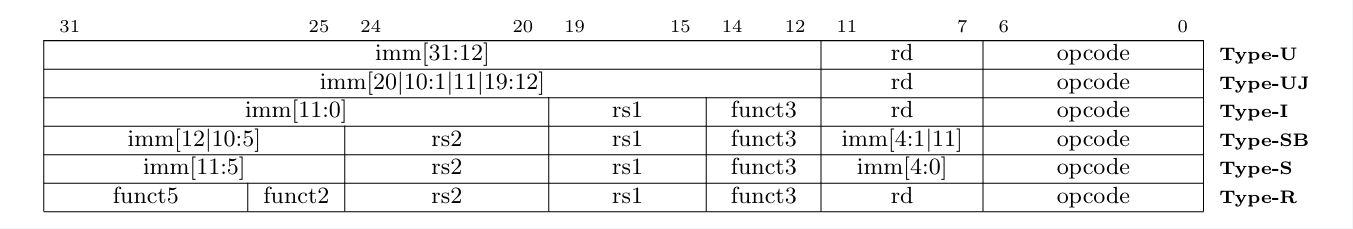


图2 指令的基本格式

在表1中，我们可以看到六种不同类型的指令：

R型指令：这一类型的指令专门用于寄存器和寄存器之间的操作，它允许程序直接对寄存器内的数据进行处理，是计算机内部数据处理的核心。

I型指令：这类指令主要处理短立即数和内存载入（load）操作。短立即数是指直接包含在指令中的小范围数值，而内存载入操作则是将数据从内存读取到寄存器中，以供后续处理。

S型指令：S型指令专门用于内存存储（store）操作，即将寄存器中的数据写回到内存中。这一操作是数据持久化的关键步骤，确保了数据在需要时可以被重新读取。

B（SB）指令：这是一种条件跳转指令，它根据一定的条件判断是否执行跳转操作。这种指令在程序流程控制中扮演着重要角色，能够实现分支结构，使得程序可以根据不同的情况执行不同的代码路径。

U型指令：U型指令用于处理长立即数操作。与I型指令中的短立即数不同，长立即数可以表示更大范围的数值，满足程序中对大数值处理的需求。

J（UJ）型指令：这是一种无条件跳转指令，它不论条件如何都会执行跳转操作。这种指令在程序控制流中用于实现循环、函数调用等结构，是程序跳转的基本手段。

为了更好地管理和实现这些指令，每一种指令类型都可以采用C++中的一个类来实现，这些类将封装指令的具体实现细节，包括指令的编码方式、执行流程以及可能涉及的数据结构等，从而提供一个清晰、统一的接口供程序其他部分调用。通过这样的设计，可以大大提高指令集的可维护性和可扩展性。