|  |  |  |  |
| --- | --- | --- | --- |
| |  | | --- | | 硬件课程设计报告 | | 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、 设计目的 4](#_Toc177917634)

[二、 设计过程 4](#_Toc177917635)

[2.1 设计过程说明 4](#_Toc177917636)

[2.1.1 模型机位数选定 4](#_Toc177917637)

[2.1.2 寄存器设计 5](#_Toc177917638)

[2.1.3 设计指令格式 5](#_Toc177917639)

[2.2 指令集设计 6](#_Toc177917640)

**表目录**

[表1 RISC-V模拟器寄存器 5](#_Toc177917069)

**图目录**

[图1 指令的格式图1 指令的格式 1 5](#_Toc177917116)

# 一、 设计目的

RISC-V是一种开源的指令集架构（ISA），具有高度的灵活性和可扩展性。RISC-V规范定义了基本整数指令集（RV32I/RV64I，分别对应32位和64位），并允许通过可选扩展来添加额外的功能，如浮点运算（RV32F/RV64F）、向量运算（RVV）、原子操作（RVA）、压缩指令集（RVC）等。这种模块化设计使得RISC-V能够根据不同的应用需求进行定制，从而优化性能、功耗和成本。

C++作为一种强大的编程语言，提供了丰富的编程特性和库支持，使得我们可以根据需要灵活地实现RISC-V模拟器的各种功能。此外，由于RISC-V架构的模块化设计，模拟器可以很容易地进行定制和扩展，以适应不同的应用场景和需求。

本次硬件课程设计，我们小组使用C++实现RISC-V模拟器目的在于：

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，你们可以尝试实现不同的RISC-V扩展指令集，如浮点运算、向量运算等，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。

# 二、 设计过程

2.1 设计过程说明

### 2.1.1 模型机位数选定

该RISC-V模型机采用32位，这一设计意味着它能够直接处理32位宽的数据和操作码，从而支持广泛的应用程序和数据类型。32位架构不仅提高了数据处理能力，还使得RISC-V模型机在保持简洁性的同时，具备了强大的计算性能和灵活性。

### 2.1.2 寄存器设计

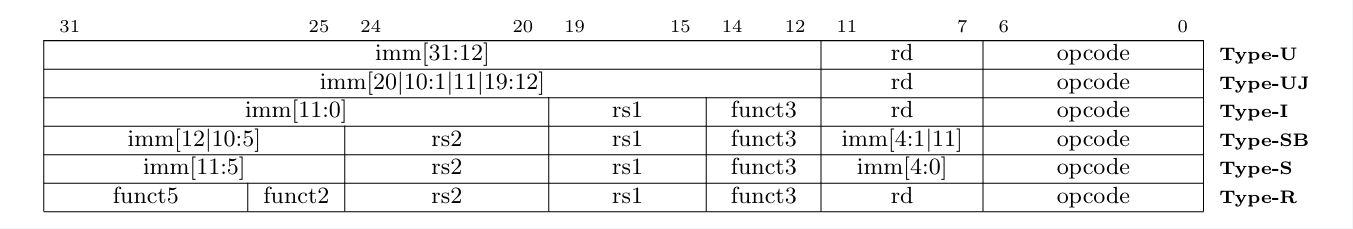
设计RISC-V模拟器需要使用的寄存器，见表1所示，RISC-V架构定义了32个32位宽的通用寄存器，这些寄存器在大多数指令中作为操作数使用。其中31个是常规寄存器，可以自由地读写，1个恒为0值的x0寄存器，只能读。

表1 RISC-V模拟器寄存器

|  |  |  |
| --- | --- | --- |
| 寄存器 | ABI名称 | 说明 |
| x0 | zero | 0值寄存器，读取数据为0 |
| x1 | ra | 用于返回地址(return address) |
| x2 | sp | 用于栈指针（stack pointer） |
| x3 | gp | 用于通用指针 (global pointer) |
| x4 | tp | 用于线程指针 （thread pointer） |
| x5 | t0 | 用于存放临时数据或者备用链接寄存器 |
| x6-x7 | t1-t2 | 用于存放临时数据寄存器 |
| x8 | s0/fp | 需要保存的寄存器或者帧指针寄存器 |
| x9 | s1 | 需要保存的寄存器 |
| x10-x11 | a0-a1 | 函数传递参数寄存器或者函数返回值寄存器 |
| x12-x17 | a2-a7 | 函数传递参数寄存器 |
| x18-x27 | s2-s11 | 需要保存的寄存器 |
| x28-x31 | t3-t6 | 用于存放临时数据寄存器 |

### 2.1.3 设计指令格式

RISC-V有六种基本指令格式，见图1所示为各种指令的格式，其中opcode :指令操作码，imm：代码立即数，funct：代表指令对应的功能，rs1：源寄存器1，rs2：源寄存器2，rd：目标寄存器

图1 指令的格式

见表2所示为具体的每种的指令实现的功能。

表2 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 |

### 2.1.4 内存设计

采用C++自带的库函数map实现，在C++中，std::map是一个非常有用的容器，它存储元素形成键值对（key-value pairs），其中每个键都是唯一的，并且根据键的排序准则自动排序元素。天然的形成了存储器的地址结构，其key值对应的是内存的地址，value存储的数据。

2.2 指令集设计

### 2.2.1 R型指令

如表3所示的为R型指令的基本格式，一共实现了9条指令，基本功能是用于寄存器和寄存器操作，如算术逻辑运算或数据计算，并将其封装为一个类名为Rtype，如图2所示为R型指令的类，为R型指令提供方便的调用接口。

表3 R型指令的格式

|  |  |  |
| --- | --- | --- |
| 指令名称 | 格式 | 功能 |
| ADD指令 | ADD rd,rs1,rs2 | rs1 + rs2 -> rd |
| SUB指令 | SUB rd,rs1,rs2 | rs1 - rs2 -> rd |
| XOR指令 | XOR rd,rs1,rs2 | rs1 ^ rs2 -> rd |
| SRL指令 | SRL rd,rs1,rs2 | (逻辑右移)rs1 >> rs2 -> rd |
| OR指令 | OR rd,rs1,rs2 | rs1 | rs2 -> rd |
| AND指令 | AND rd,rs1,rs2 | rs1 & rs2 -> rd |
| SLL指令 | SLL rd,rs1,rs2 | (逻辑左移)rs1 << rs2 -> rd |
| SLT指令 | SLT rd,rs1,rs2 | 有符号数进行比较，若前者小于后者，那么将1保存到地址为rd的通用寄存器 |
| SRA指令 | SRA rd,rs1,rs2 | (算术右移)rs1 >> rs2 -> rd |



图2 R型指令类

### 2.2.2 I型指令

如表4所示的为I型指令的基本格式，一共实现了9条指令，基本的功能是用于短的立即数和内存载入指令load操作，如从内存中去除数据和对立即数的算术逻辑运算或数据计算，并将其封装为一个类名为Itype，如图3所示为I型指令的类，为I型指令提供方便的调用接口。

表4 I型指令的格式

|  |  |  |
| --- | --- | --- |
| 指令名称 | 格式 | 功能 |
| LB 指令 | LB rd,rs1,imm | M[rs1 + imm] -> rd  从存储器种读出1个字节的数据按符号位扩展后，送入rd |
| LH 指令 | LH rd,rs1,imm | M[rs1 + imm] -> rd  从存储器种读出2个字节的数据按符号位扩展后，送入rd |
| LW 指令 | LW rd,rs1,imm | M[rs1 + imm] -> rd  从存储器种读出4个字节的数据按符号位扩展后，送入rd |
| LBU 指令 | LBU rd,rs1,imm | 取无符号1字节 |
| LHU 指令 | LHU rd,rs1,imm | 取无符号2字节 |
| ADDI 指令 | ADDI rd,rs1,imm | rs1 + imm -> rd |
| XORI 指令 | XORI rd,rs1,imm | rs1 ^ imm -> rd |
| ANDI 指令 | ANDI rd,rs1,imm | rs1 & imm -> rd |
| ORI 指令 | ORI rd,rs1,imm | rs1 | imm -> rd |

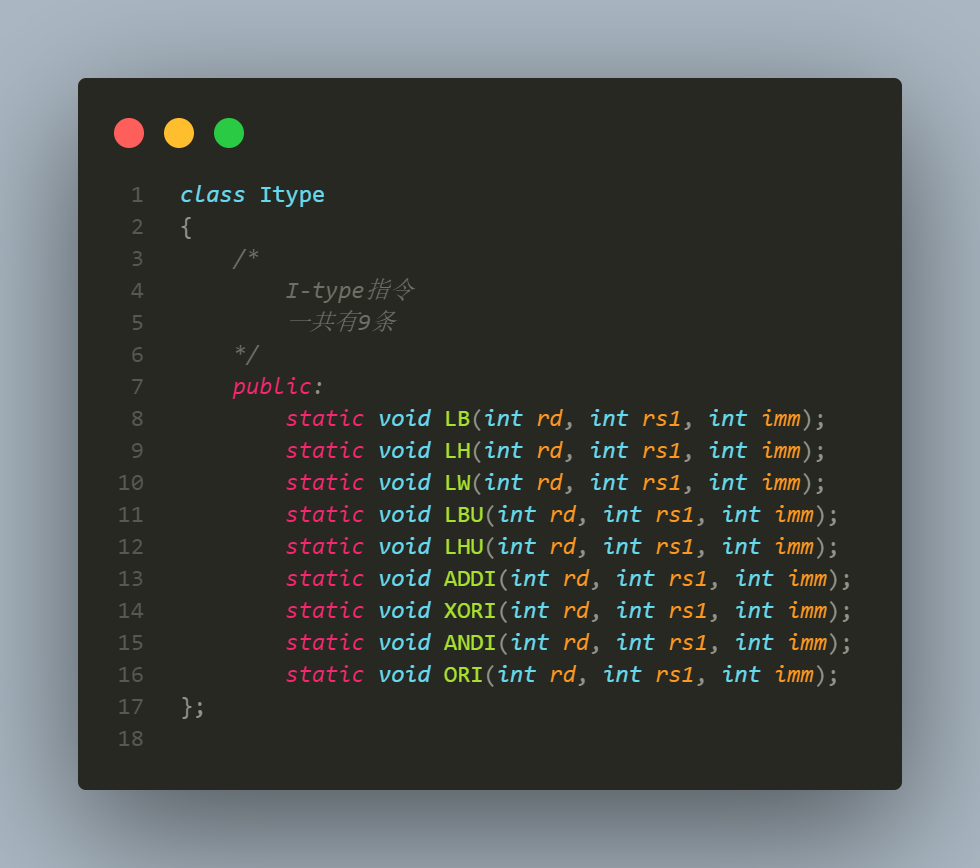


图3 I型指令类

### 2.2.3 S型指令

如表5所示的为S型指令的基本格式，一共实现了3条指令，基本的功能是用于内存存储store操作，如存储数据，并将其封装为一个类名为Stype，如图4所示为S型指令的类，为S型指令提供方便的调用接口。

表5 S型指令的格式

|  |  |  |
| --- | --- | --- |
| 指令名称 | 格式 | 功能 |
| SB 指令 | SB rs1,rs2,imm | rs1 -> M[rs2 + imm]  存储字节，将通用寄存器里的低[7:0]位，存储到特定的地址单元 |
| SH 指令 | SH rs1,rs2,imm | rs1 -> M[rs2 + imm]  存储字节，将通用寄存器里的低[15:0]位，存储到特定的地址单元 |
| SW 指令 | SW rs1,rs2,imm | rs1 -> M[rs2 + imm]  存储字节，将通用寄存器里的低[31:0]位，存储到特定的地址单元 |



图4 S型指令类

### 2.2.4 B型指令

如表6所示的为B型指令的基本格式，一共实现了6条指令，基本的功能是用于有条件跳转操作，并将其封装为一个类名为Btype，如图5所示为B型指令的类，为B型指令提供方便的调用接口。

表6 B型指令的格式

|  |  |  |
| --- | --- | --- |
| 指令名称 | 格式 | 功能 |
| BEQ 指令 | BEQ rs1, rs2, imm | rs1和rs2相等，则跳转PC+imm |
| BNE 指令 | BNE rs1, rs2, imm | rs1和rs2不相等，则跳转PC+imm |
| BGE 指令 | BGE rs1, rs2, imm | rs1大于等于rs2，则跳转PC+imm |
| BLT 指令 | BLT rs1, rs2, imm | rs1小于rs2，则跳转PC+imm |
| BLTU 指令 | BLTU rs1, rs2, imm | 无符号rs1小于无符号rs2，则跳转PC+imm |
| BGEU 指令 | BGEU rs1, rs2, imm | 无符号rs1大于等于无符号rs2，则跳转PC+imm |



图5 B型指令类

### 2.2.5 U型指令

如表7所示的为U型指令的基本格式，一共实现了2条指令，基本的功能是用于长立即数操作，并将其封装为一个类名为Utype，如图6所示为U型指令的类，为U型指令提供方便的调用接口。

表7 U型指令的格式

|  |  |  |
| --- | --- | --- |
| 指令名称 | 格式 | 功能 |
| LUI 指令 | LUI rd, imm | 一个20位的立即数加载到寄存器rd的高20位，低12位为0 |
| AUIPC 指令 | AUIPC rd, imm | 将一个20位立即数加到PC的高20位，生成一个32位地址 |

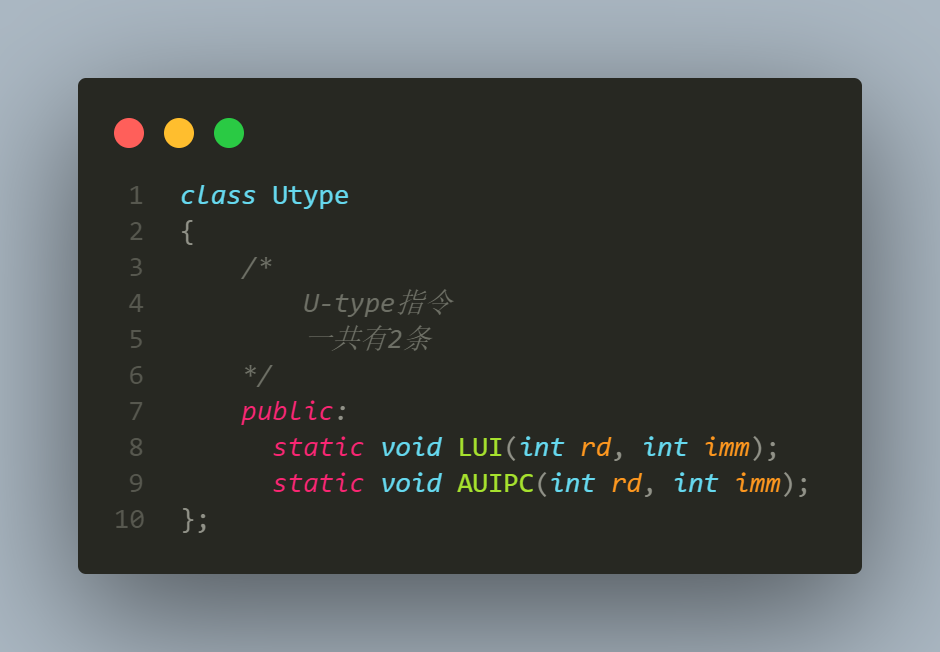


图6 U型指令类

该指令作用在于存取全局变量的地址，如图7所示，由于是32位机，若PC为20位，那么可以通过使用AUIPC指令进行数据左移，将该20位数据放入高20位。

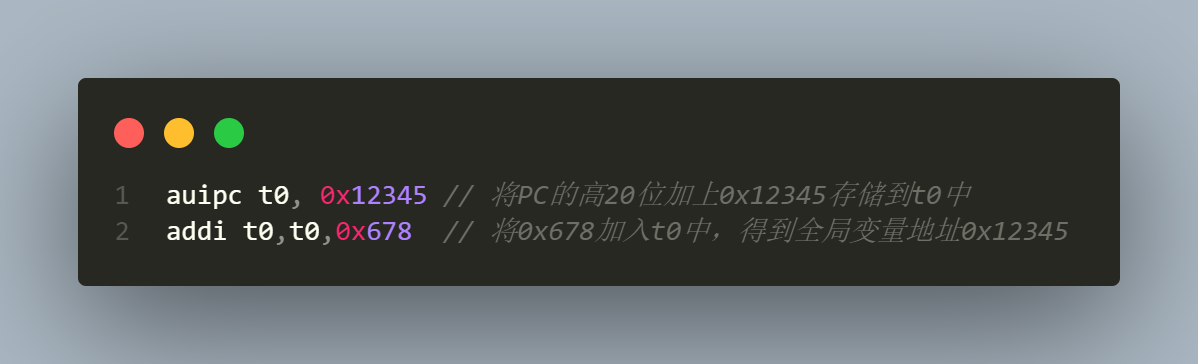


图7 AUIPC举例

### 2.2.5 J型指令

如表8所示的为J型指令的基本格式，一共实现了1条指令，基本的功能是用于无条件跳转操作，并将其封装为一个类名为Jtype，如图8所示为J型指令的类，为J型指令提供方便的调用接口。

表8 J型指令的格式

|  |  |  |
| --- | --- | --- |
| 指令名称 | 格式 | 功能 |
| JAL指令 | JAL rd, imm | 将PC的值紧随其后的那条指令地址，存入rd寄存器中，然后将PC=PC+imm |

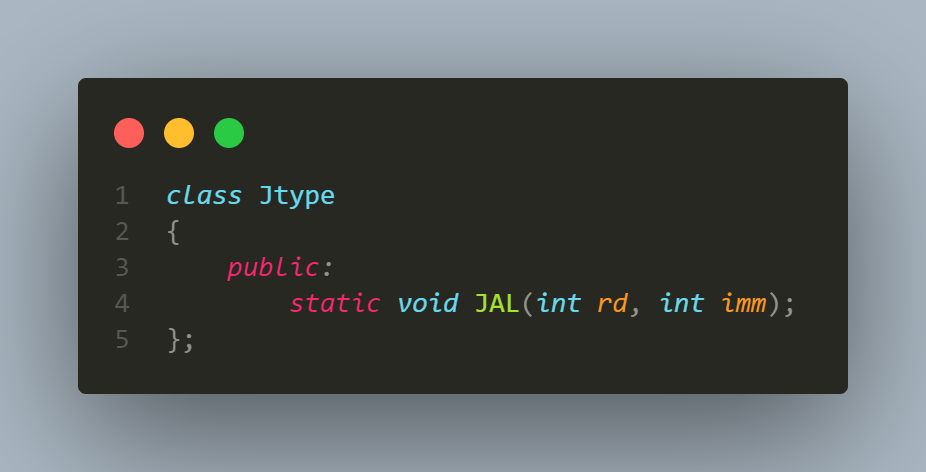


图8 J型指令类

2.3 **代码数据处理**

### 2.3.1 指令文件读取

首先，给出文件的所在路径，利用C++的文件流进行文件的读写，读入后去除每一条指令的前导空格和空行，有以下四种情况：

1. 正常的一条指令，直接存入数组中即可。
2. 当前行有注释，注释使用“#”标识。首先找到“#”所在的位置，然后截取剩余部分，检查是否剩余部分有指令，若有，则将指令存入数组。
3. 标号与指令不在同一行。读取标号，并将标号所在的PC值存入标号数组， 以供后序的跳转指令的使用。
4. 标号与指令在同一行。同样读取标号，然后通过“：”的位置截取指令，存入数组中。

在处理这四种情况的时候，同时计算PC值，由于是32位机，每一次加“4”指向下一条指令，计算PC有利于整个程序的运行，是整个程序运行的关键。

### 2.3.2 模拟器初始化

寄存器：

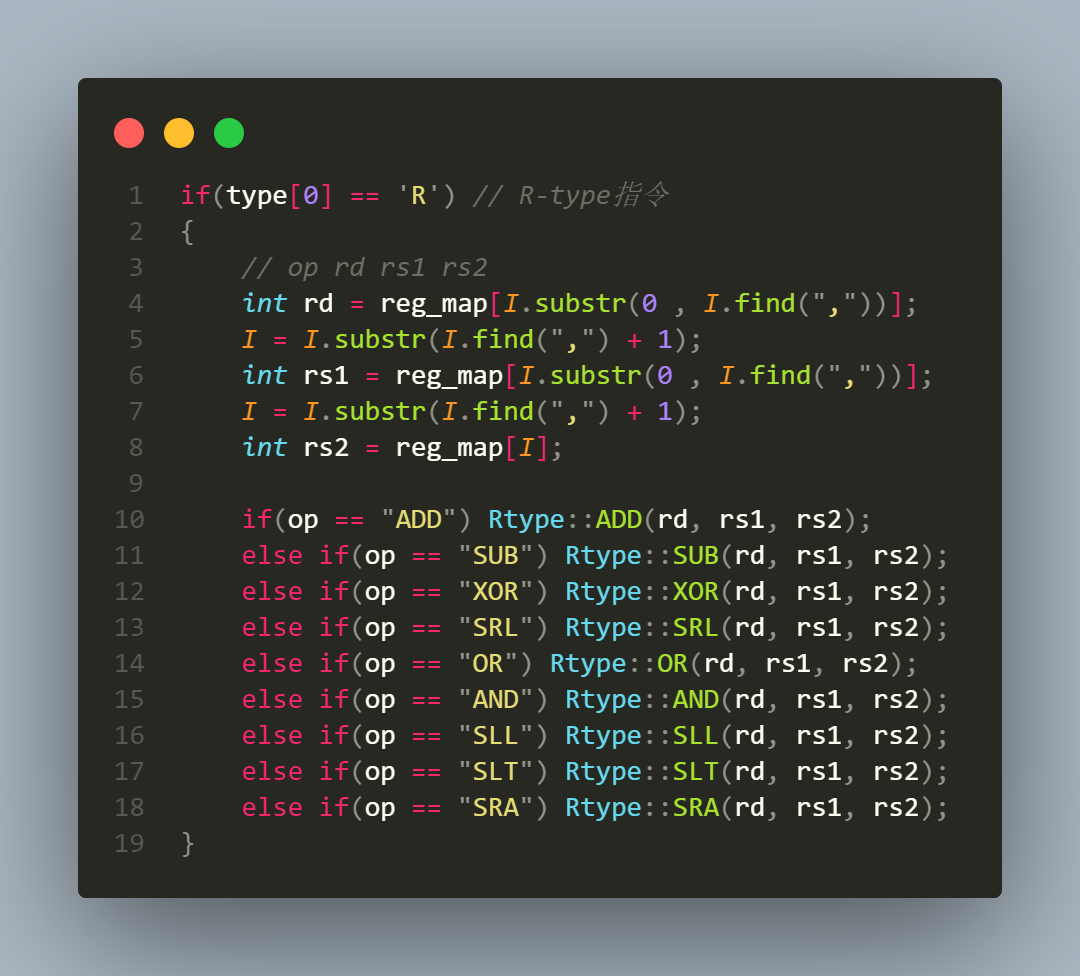
1. RISC-V 32个通用寄存器：采用顺序表，数据结构存储，一开始全部初始化为0
2. 寄存器名到寄存器编号的映射：采用哈希表(unordered\_map)，这种数据结构，目的在于可以使用ABI名称直接对寄存器访问，方便代码的编写。
3. 寄存器编号到寄存器名的映射：采用哈希表(unordered\_map)，这种数据结构，目的在于最后可以通过编号直接输出对应的寄存器名，方便最后结果的输出。

存储器：采用哈希表(map)，这种数据结构，目的在于地址递增的存储数据，从低地址往高地址存数，并采用小端方式进行存储，即高位存放在高字节。

标号：采用哈希表(unordered\_map)，这种数据结构，目的在于条件跳转的时候，PC可以快速的找到需要运行的程序的位置。

### 2.3.3 运行指令

通过使用PC指向下一条需要执行的指令，六种不同的类型采用六种不同的分析方法，由于不同类型对应的格式不同，因此需要分类分析。每一条指令，无论什么类型都是操作码和操作数组成，对于操作码，本模拟器对操作码的大小写不敏感，即大小写都可以作为操作码。如图9所示即为R型指令的的分析过程，取出每一条指令的操作数，调用类的函数最终获得想要的结果。



2.3整机调试

2.3.1测试程序设计

（根据指令集设计程序，格式参考教材P90的表6-7，给出类汇编代码及内存中保存的16进制代码ram.MIF截图）

2.3.2 仿真结果

（可通过标注、示意等方法，显示各条指令及其结果）

2.3.3 下载后执行结果

（模式选择+管脚绑定截图+输出结果说明）

三、实验总结及心得体会

学生评价-问卷调查（学生填写）

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | 完全同意 | 同意 | 基本同意 | 不同意 | 完全不同意 |
| 课程目标1 | 培养学生系统设计能力，使学生掌握基本计算机的设计能力，重点掌握CPU的设计。培养学生独立完成计算机各个模块的设计，掌握各模块间的关系及作用。培养学生对数据流、地址流、控制流的设计能力，总线形式的选择及设计能力。 |  |  |  |  |  |
| 课程目标2 | 培养学生各类工程设计能力，包括逻辑电路设计能力、硬件描述语言的程序设计能力，以及电路图与程序语言配合的设计方法等。在QuartusⅡ环境下，完成模型机的逻辑设计，该模型机需符合RISC机型的特点并能够正常工作。 |  |  |  |  |  |
| 课程目标3 | 培养学生掌握多种工程测试手段，包括仿真指令测试、仿真程序测试、以及实验仪上脱机程序测试的方法，进行整机调试，以验证模型机设计的完整性。培养学生独立思考、查找资料、设计、修改、调试的能力；不设上限的开放性设计目标，促使学生不断完善和提高。 |  |  |  |  |  |