**计组lab1实验报告——单周期CPU电路设计**

**实验人：谢志康**

**学号：22307110187**

**实验完成时间：2024.3.12**

**目录：1.学习参考资料 2.模块介绍 3.功能介绍举例**

**4.单独模块仿真测试 5.文件夹说明**

1. **学习参考资料**

RISC-V相关指令集以及架构参考学习——

[RISC-V手册 (gitlab.com)](https://gitlab.com/fudan-systa/arch-2023spring-fudan/-/wikis/uploads/RISC-V-Reader-Chinese-v2p1.pdf)

1. [初识RISC-V - Wahahahehehe - 博客园 (cnblogs.com)](https://www.cnblogs.com/wahahahehehe/p/15574316.html)

实现64位RISC-V架构CPU，使用RV64I指令集，按照实验手册要求实现了addi xori ori andi lui jal beq add sub and or xor auipc jalr这些指令（以及考虑到以后加入memory等模块，访存相关的一些指令，简要写了一点）

[RISC-V 指令格式和6种基本整数指令\_csrrw-CSDN博客](https://blog.csdn.net/qq_39507748/article/details/120150936)

设计参考——

[RISC-V CPU 设计（6）： RV64I CPU 控制器模块设计思路与实现 - 泰晓科技 (tinylab.org)](https://tinylab.org/cpu-design-riscv-cpu-controller-module-design/)

1. **模块介绍**

采用verilog语言编写（也有一版systemverilog语言，尝试与测试环境接起来的版本，我一同放在压缩文件中了）共有defines.vh（预定义的头文件），NPC、PC、SEXT、Control、RegFile、RegFile\_wD\_MUX、ALU、ALU\_input\_MUX、MEM这几个组件模块，以及myCPU这个顶层模块将各组件相接。

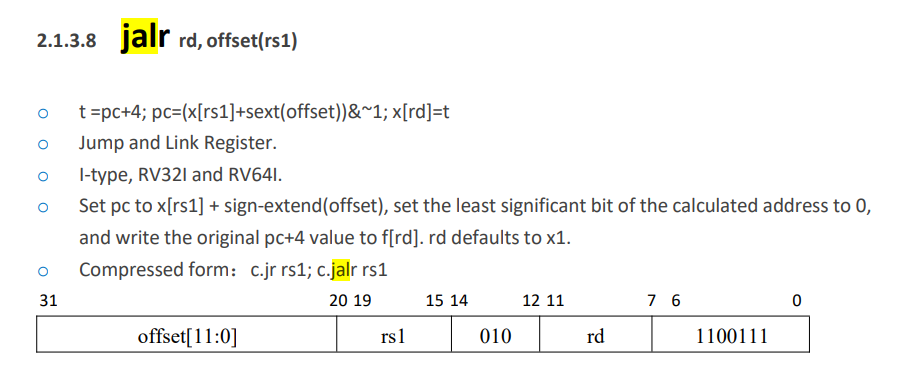
* 1. defines.vh：预定义一些编码，如将各个alu操作的funct3、funct7、opcode编码定义为具体名称（如`define FUNCT7\_SUB 7'b0100000）为了提供便利。
  2. NPC：next pc，给出下一时钟周期的PC值。Input有PC（当前时钟周期的指令地址）、offset（立即数的偏移量（经过符号扩展））、alu\_c（来自ALU的运算结果）branch （判断是否发生跳转的使能）、npc\_op（决定NPC的最终取法，是顺序执行下一条指令PC+4还是取PC+offset还是直接置为alu\_c），output有npc（下一条指令的地址）
  3. PC：负责保存当前时钟周期的指令地址。接收来自NPC模块的指令地址，当时钟上升沿到来时，输出指令地址给指令寄存器IROM读取。
  4. SEXT：根据不同指令类型进行立即数扩展。
  5. Control：根据指令产生对应的控制信号。 拆解指令为opcode，funct3，funct7等值（若有）产生sext\_op（对立即数按不同的指令类型进行符号扩展的控制信号）、npc\_op（产生下一个时钟周期的PC值的控制信号）、alu\_op（ALU模块的控制信号）、alua\_sel（ALU操作数A的选择信号）、alub\_sel（ALU操作数B的选择信号）、rf\_we（寄存器堆的写使能信号）、rf\_wsel（寄存器堆写回的选择信号），还有目前没有要求也未完全实现的DRAM写信号（根据字、半字等等）

要求指令中lui、auipc、jalr、jal四个指令均在该模块进行了设置（由其它模块做对应的运算）后面会举例解释jalr的执行过程。

* 1. RegFile和RegFile\_wD\_MUX：RegFile\_wD\_MUX是一个多选器，根据rf\_wsel的值来给出写回的数据选择。RegFile即寄存器堆（内含初始化与reset操作）。根据位置读写（写要有写使能）。
  2. ALU和ALU\_input\_MUX：ALU\_input\_MUX是一个多选器，根据alua\_sel和alub\_sel的值来选择源操作数，output A、B返回回去给ALU计算。ALU即计算模块了，在这里实现了剩余要求的addi xori ori andi beq add sub and or xor指令（至于是add还是addi这种在mux中决定好的）此外还顺带实现了bne、blt、各个shift指令等等。
  3. MEM模块进行内存管理（后续在测试框架接dbus即可，目前并没有用到）
  4. IROM模块存放指令（后续在测试框架接ibus即可，目前并没有用到）
  5. myCPU：顶层模块，将各模块链接起来。

1. **功能介绍举例**

对于各种指令，我是开了很多case来选择执行，这样比较便于管理且便于以后加指令。以jalr的执行逻辑作为例子我来讲讲：在Control模块进入`OPCODE\_JALR条件执行jalr指令（sext选择i型指令扩展，npc选择以alu计算结果来更新，两个源操作数分别选择rd1和sext，设置寄存器写使能为PC4，alu操作选择为add）——在RegFile模块中，rD1选择的是rR1，rR1在顶层文件中传入的是inst[19:15]——在RegFile\_wD\_MUX模块中，要写入的数据wD设为PC+4（由于control中使能设为的PC4）在RegFile中写入——在ALU中实现对应add操作，两个源操作数在ALU\_input\_MUX中分别设置为了reg[inst[19:15]]和sext（offset的i型扩展）相加作为alu\_c的值（alu模块的计算结果）——最后，在NPC中，由于Control设置的npc使能为`NPC\_SEL\_ALU，因此npc=alu\_c & ~1更新了PC值。



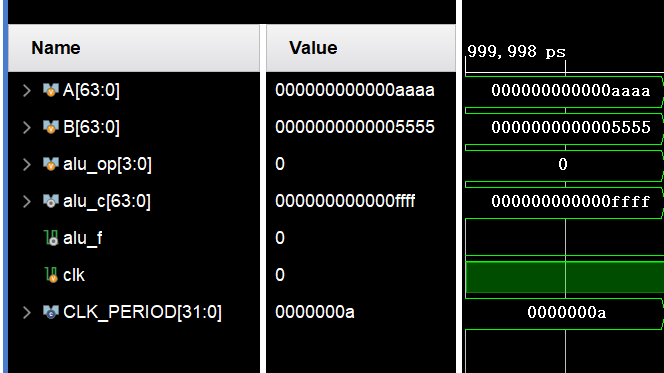
遵从jalr指令步骤执行。

1. **单独模块仿真测试**

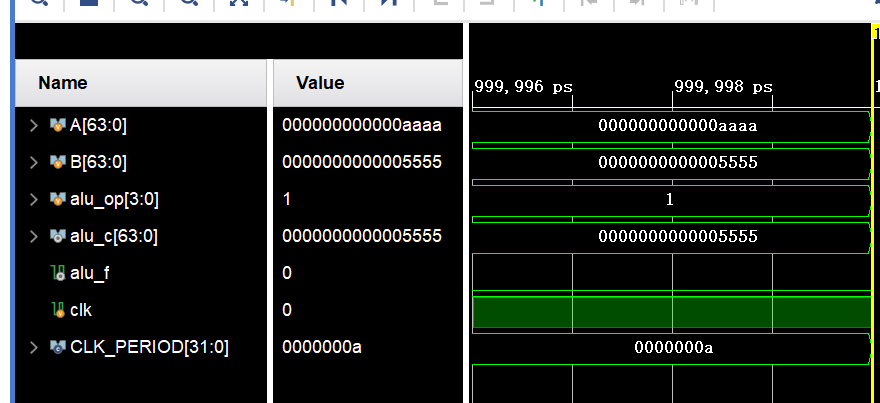
整体接入测试框架暂未实现，当前仅仅编写了对每个组件模块的功能进行测试的仿真文件。以ALU模块为例：

输入的AB由ALU\_input\_MUX决定，alu\_c即运算结果。

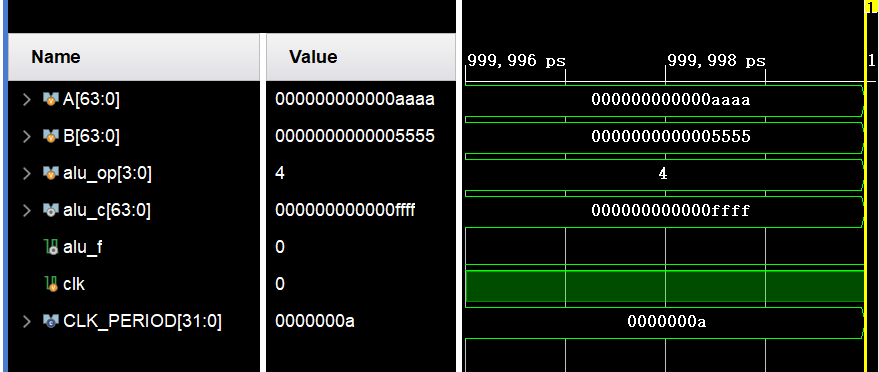
Add功能：



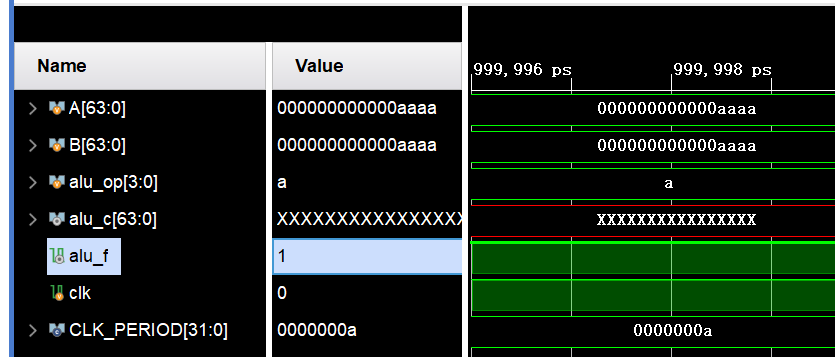
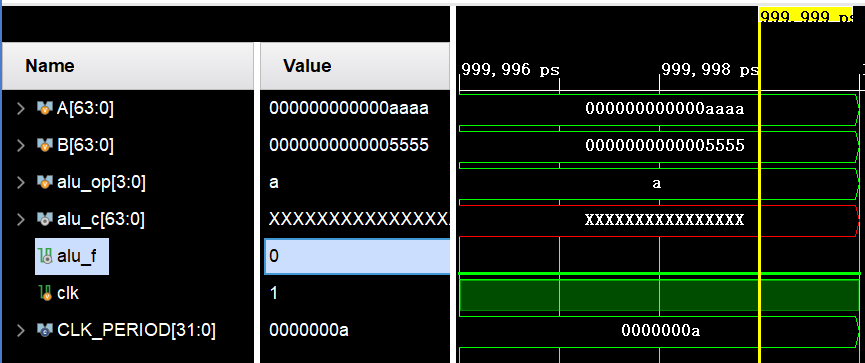
Sub功能：



Xor功能：



BEQ功能： （A!=B 不跳转，aluf=0 A==B跳转，aluf=1）



对于jalr、jal、auipc、lui指令，正如part3介绍的一样，只是由control模块给出相应的select，然后由各个mux模块选择对应的数据，再交给alu进行计算，并对RegFile做出相应操作。

其它组件也都进行了单独的功能测试，这里不一一列举了。

1. **文件夹说明**

最开始我是基于verilog语言开发，见同级文件夹**RV64\_Single\_Period\_CPU**项目，助教只用检查这个文件夹代码即可。

另一个**arch-2024-main**文件夹，src内的文件和这个项目是一样的，只是为了接起助教给的测试框架，我将整体代码改成了systemverilog语言，并将原来的irom模块和memory模块删除，尝试接上ibus和dbus。正确性并没有经过检验，唐思源助教说后续才会教测试框架的用法，现在不用用那个框架测试。

**CPU流程图.pdf**即我的CPU的各个框架逻辑联系流程图。

**report.pdf**即本文，实验报告。