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)

第1章　初识 RISC-V - Wahahahehehe - 博客园 (cnblogs.com)

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

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)

2. 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。

⑥ 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）。

⑦ 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。

⑧ 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）

⑨ 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）

⑩ myCPU：顶层模块，将各模块链接起来。

3. 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。

2.1.3.8 **jalr** rd, offset(rs1)

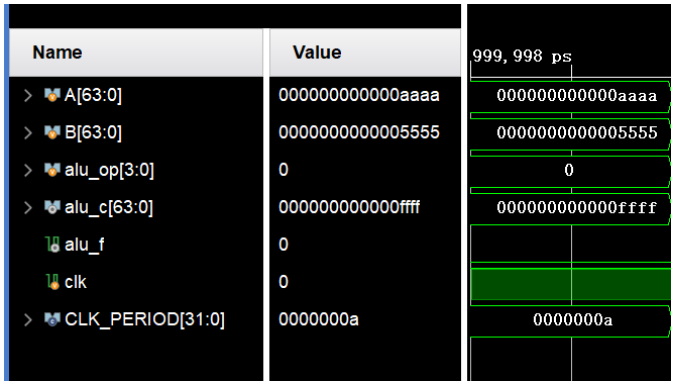
- t =pc+4; pc=(x[rs1]+sext(offset))&~1; x[rd]=t
- Jump and Link Register.
- I-type, RV32I and RV64I.
- Set pc to x[rs1] + sign-extend(offset), set the least significant bit of the calculated address to 0, and write the original pc+4 value to f[rd]. rd defaults to x1.
- Compressed form： c.jr rs1; c.jalr rs1

31 20	19 15	14 12	11 7	6 0
offset[11:0]	rs1	010	rd	1100111

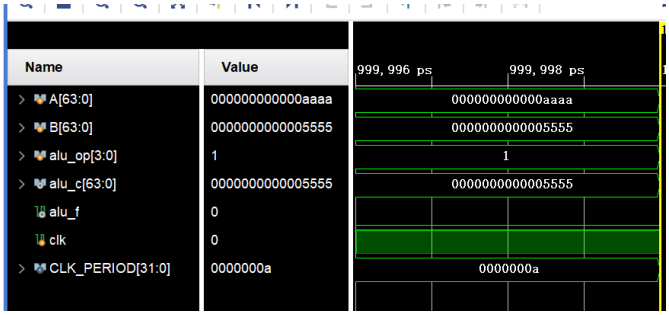
遵从 jalr 指令步骤执行。

4. 单独模块仿真测试

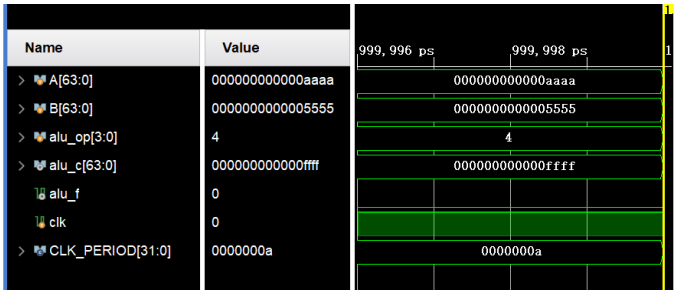
整体接入测试框架暂未实现，当前仅仅编写了对每个组件模块的功能进行测试的仿真文件。
以 ALU 模块为例：
输入的 AB 由 ALU_input_MUX 决定，alu_c 即运算结果。
Add 功能：



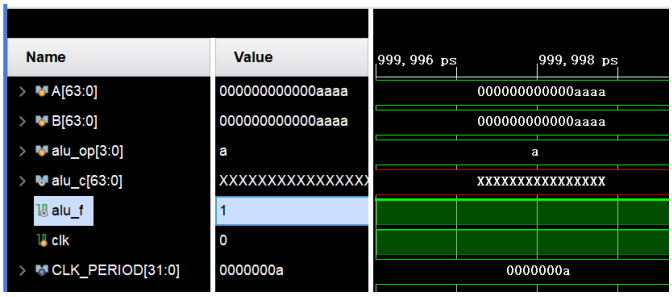
Sub 功能：



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。

5. 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。