中国科学院大学
《计算机组成原理(研讨课)》实验报告

姓名 朱夏楠 学号 2022K8009929031 专业 计算机科学与技术

实验项目编号 4 实验名称 定制 RISC-V 功能型处理器设计

注 1:	撰写此 Word 格式实验报告后以 PDF 格式保存 SERVE CloudIDE 的 `/home/serve-ide/cod-lab/reports` 目录下 (注意: reports 全部小写)。文件命名规则: `prjN.pdf`, 其中 `prj` 和后缀名 `pdf` 为小写, `N` 为 1 至 4 的阿拉伯数字。例如: `prj1.pdf`。PDF 文件大小应控制在 5MB 以内。此外, 实验项目 5 包含多个选做内容, 每个选做实验应提交各自的实验报告文件, 文件命名规则: `prj5-projectname.pdf`, 其中 "-" 为英文标点符号的短横线。文件命名举例: `prj5-dma.pdf`。具体要求详见实验项目 5 讲义。
注 2:	使用 `git add` 及 `git commit` 命令将实验报告 `PDF` 文件添加到本地仓库 master 分支, 并通过 `git push` 推送到实验课 SERVE GitLab 远程仓库 master 分支(具体命令详见实验报告)。
注 3:	实验报告模板下列条目仅供参考, 可包含但不限定如下内容。实验报告中无需重复描述讲义中的实验流程。

一、 逻辑电路结构与仿真波形的截图及说明(比如 Verilog HDL 关键代码段 {包含注释} 及其对应的逻辑电路结构图 {自行画图, 推荐用 PPT 画逻辑结构框图后保存为 PDF, 再插入到 LaTeX. 中}、相应信号的仿真波形和信号变化的说明等)

• RISC-V 处理器设计。

本次 RISC-V 处理器是基于定制 MIPS 功能型处理器改进而来,与 MIPS 的主要区别在于译码逻辑的改变,立即数形式增多,删减和增加了一些新的指令,以及放弃了分支延迟槽的设计。在实验中实现的 MIPS 指令,除了 RType 指令由 funct 段决定,其余类均由 opcode 决定,而 RISC-V 的各类指令由 opcode 决定,同一类的各种指令由 funct3 和 funct7 决定;此外 RISC-V 相对 MIPS,立即数的形式有了较大的改变,减少了立即数在指令中所占的长度,增加了立即数的种类。与 MIPS 相比,这种设计保证了各类指令之间相同的段更多,同时立即数被安排的位置相同,便于在译码阶段就能生成指令所需的 32 位立即数,经过一定的设计,将各段拼接生成立即数所需的门电路也会更少; RISC-V 相对 MIPS 删除了一些指令, 如非对齐存取数据, 这些指令在设计时会明显的增加逻辑电路的复杂度, 而观察了实验中 benchmark 的汇编代码后发现, 很少使用非对齐存取指令, 相对的, 这种非对齐取数据带来的便利并不显著,我认为这种删除是合理的。

基于以上几点设计 RISC-V 处理器,译码逻辑的改变意味着需要修改控制单元对控制信号的赋值逻辑,同时 ALU 的操作码也可以适当修改,在 MIPS 中,RType 指令由 funct 决定,而 IType 的运算指令由 opcode 决定,导致 ALUop 的译码较为复杂,而 RISC-V 中,除了 RType 的减法,其余指令都可与 func3 段一一对应,因此可直接将 ALUop 与 func3 对应,这样就减少了 ALU 部分的译码复杂度。

```
//MIPS的ALUop
assign ALUopcode = ({func[1],2'b10} & {3{isAddSub}})
 | ({func[1],1'b0,func[0]} & {3{isLogicalOp}})
 | ({~func[0],2'b11} & {3{isComp & ~isjalr}})
 | ({opcode[1],2'b10} & {3{isAddSubi}})
 | ({opcode[1],1'b0,opcode[0]} & {3{isLogicalOpi}})
 | ({~opcode[0],2'b11} & {3{isCompi}})
 | (3'b010 & {3{isjalr}})
```

```
    | (3'b111 & {3{aluCOMP & ~(isbeq | isbne)}})
    | (3'b110 & {3{aluCOMP & (isbeq | isbne)}})
    | (3'b010 & {3{aluADD}});

  //RISC-V的ALUop
  assign ALUop = (funct3 & {3{(isRtype | isItype_C) & current_state[4]}}) | {2'b0,(isRtype &
      funct7[5] & current_state[4])}
            | ({~funct3[2],funct3[2],funct3[1]} & {3{isBtype & current_state[4]}});
```

此外，立即数增多导致需要对立即数扩展部分的逻辑变得更加复杂，而与之相对的，不需要在后续对立即数进行移位或者额外生成新的立即数的操作。对于直接跳转和条件跳转指令来讲，执行阶段 PC 值的更新逻辑也更加简单。

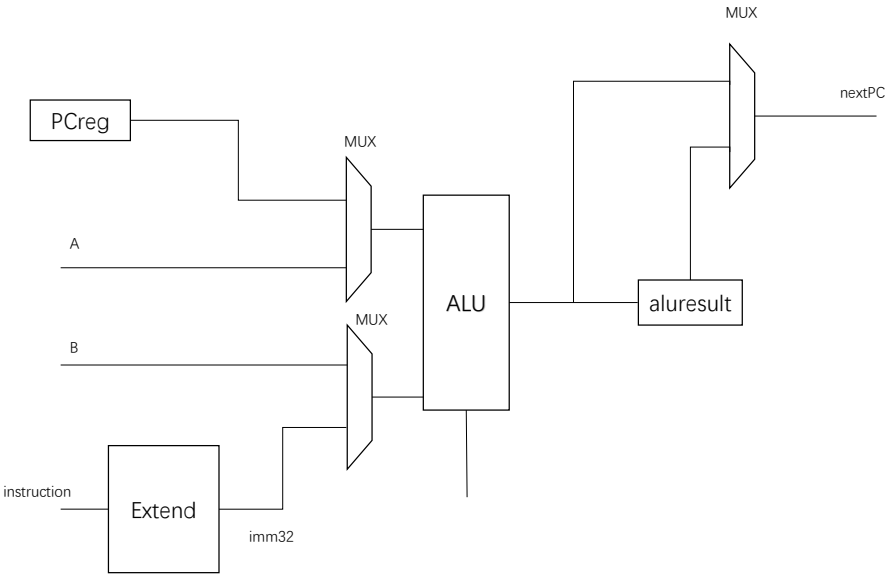

图 1: RISC-V 跳转指令的数据通路 (不含取指和读寄存器)

RISC-V 放弃了分支延迟槽的设计，仅就多周期处理器来说，由于在取值结束后就会对 PC 值进行加 4 的操作，MIPS 处理器由于分支延迟槽的存在，涉及到 PC 的指令大多需要 PC+4，因此可以方便的利用更新后的 PC 值，但对于 RISC-V 处理器，需要用寄存器保存原先的 PC 值，在运算时使用，相对而言数据通路更加复杂。

二、 实验过程中遇到的问题、对问题的思考过程及解决方法（比如 RTL 代码中出现的逻辑 bug，逻辑仿真和 FPGA 调试过程中的难点等）

- **RISC-V 处理器设计问题**。

该实验项目相较于之前的 MIPS 处理器出现较大改动的地方为 ALUop 的编码，在修改编码的过程中，我未对条件跳转指令进行特殊考虑，导致 ALUop 在条件跳转时出现错误。

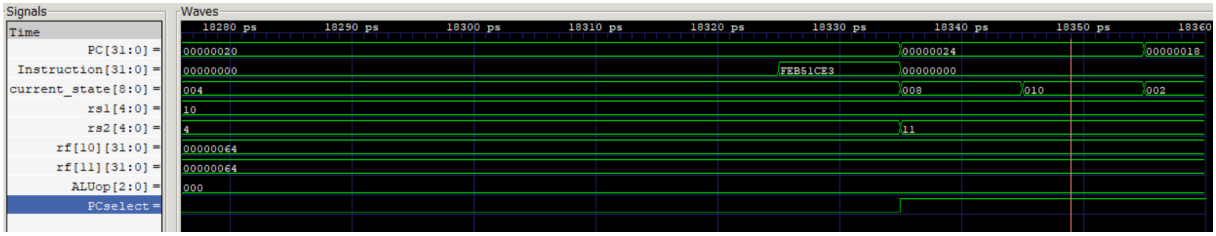

图 2: 错误波形

如图所示,在 PC 值为 0x20 时,对应指令为 bne,比较的两个寄存器分别为 10 号寄存器和 11 号寄存器,此时两寄存器中的值均为十进制下的 100,下一条指令不跳转,但执行阶段代表 PC 跳转的信号 PCselect 拉高,这是由于此时的 ALUop 错误,ALU 进行的是加法操作,而非比较操作。

由于将 ALUop 与 funct3 段一一对应，执行条件跳转指令时，ALU 在执行阶段需要进行有符号数比较或无符号数比较，两者的 ALUop 编码分别为 010 和 011。观察条件跳转指令的 funct3 段，发现需要进行无符号数比较的指令 func3 的第 2 位都为 1，因此设计条件跳转类指令对应 ALUop 编码为 $ALUop = \{\sim funct3[2], funct3[2], funct3[1]\}$。

三、 对讲义中思考题(如有)的理解和回答

- **MIPS 和 RISC-V 比较**。

通过性能计数器,可以获取 MIPS 处理器和 RISC-V 处理器在执行相同的程序时的一些数据,主要可依据执行的指令数对比二者的差异,首先可以注意到对于大部分的测试程序,MIPS 的总执行指令数都多与 RISC-V,但并不是所有的程序都是如此。

猜测有三个影响因素,一是编译器优化问题,MIPS 和 RISC-V 编译生成的汇编代码质量,直接影响程序执行所需的指令数;二是指令功能问题,在我们所实现的处理器中 MIPS 较于 RISC-V 多出了一些不太常用但在特定情境下较为方便的指令,但这类指令较为少见,因此该因素影响可能较小;三是指令集设计,MIPS 采用了分支延迟槽设计,而在我们的项目中生成对应汇编代码时,在所有分支延迟槽位置放置的指令都为 nop,这会很显著的增加程序的指令数。

四、 实验所耗时间

在课后,你花费了大约____4____ 小时完成此次实验。