中国科学院大学
《计算机组成原理(研讨课)》实验报告

姓名 宋俊仪 学号 2023K8009929018 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、 RISC-V 多周期处理器设计分析

- 状态转移分析

讲义中的多周期状态转移设计如下:(见图 1)

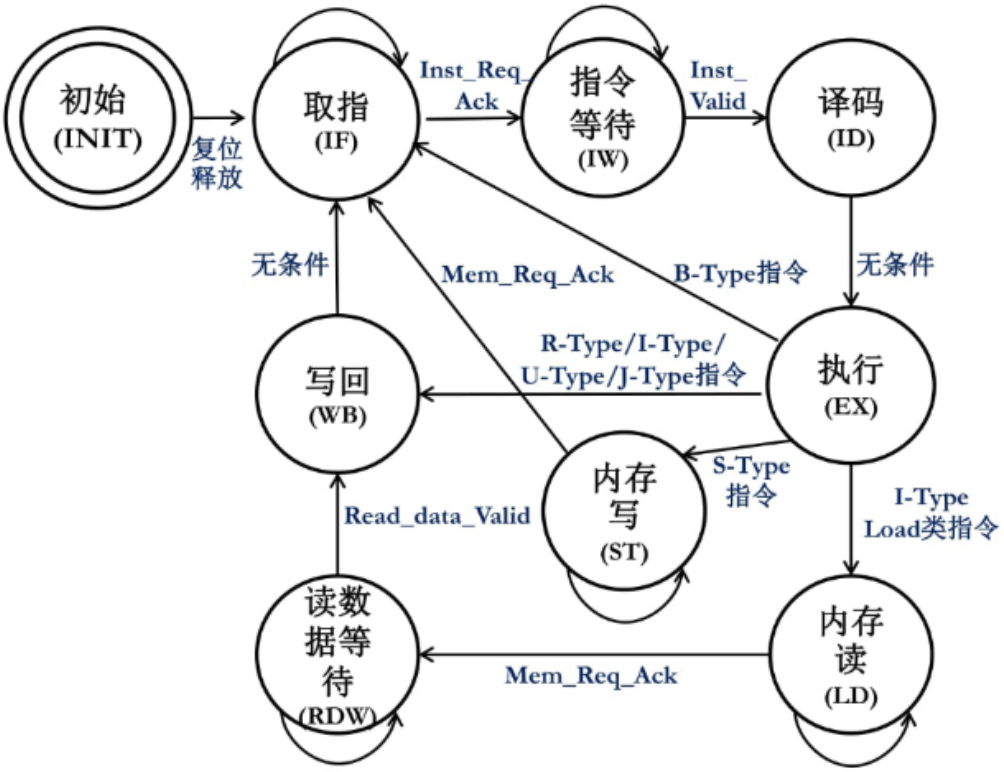

图 1: RISC-V 处理器状态转移图

通过状态转移图可以发现,RISC-V 多周期处理器和 MIPS 多周期处理器的状态转移图是非常相似的。它们都包含了取指(IF)、译码(ID)、执行(EX)、访存(MEM)和写回(WB)等状态。不同点仅仅是在各自的指令种类

有所不同，在译码阶段产生的控制信号也有所不同。除此之外，并无较大差异，因此这里不再单独赘述各个状态的功能和作用，这些分析已经在 Prj3 报告中给出。

• 逻辑电路具体设计

本次实验本没有单独设计新的逻辑电路，因为我分析实验要求，以及 RISC-V 多周期处理器的状态转移图后，发现 RISC-V 多周期处理器的逻辑电路设计和 MIPS 多周期处理器的逻辑电路设计是非常相似的。因此我直接使用了 Prj3 中单周期 CPU 的逻辑电路设计（见图 2）。

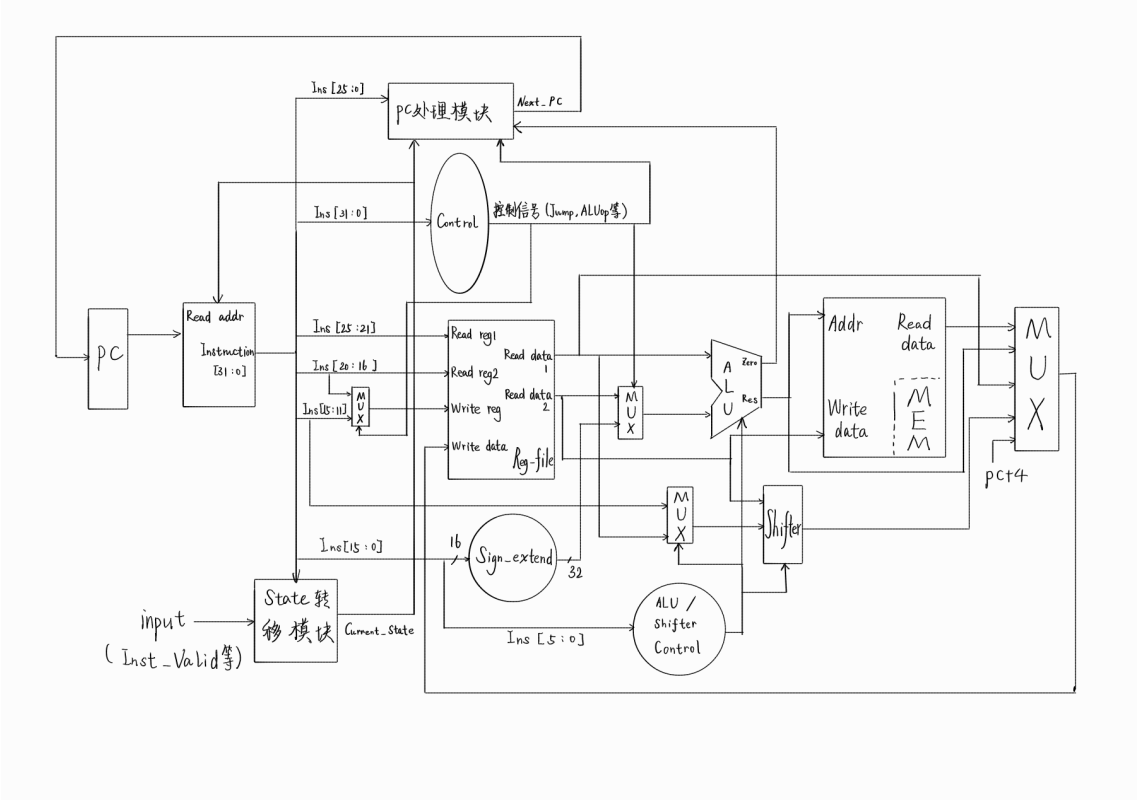

图 2: Custom_CPU 电路设计简图

• RISC-V 指令译码分析

通过查阅相关资料和对 RISC-V 指令集的分析，我发现 RISC-V 指令集的指令格式和 MIPS 指令集的指令格式是非常相似的。RISC-V 指令集也包含了 R 型、I 型、S 型、B 型、U 型和 J 型等指令格式。我依旧按照我在 Prj3 实验中我的设计思路，先完成了 RISC-V 的指令集译码表设计（见图 3）。

指令类型	序号	指令名称	指令码						指令操作
			funct7	rs2	rs1	funct3	rd	opcode	
U-type	1	LUI					5bit rd	0110111	将imm存入rd寄存器
	2	AUIPC						0010111	将PC+imm存入rd寄存器
J-type	3	JAL						1101111	将PC+4存入rd寄存器，并且跳转到PC+imm地址
I-type	4	JALR			5bit rs1	000		1100111	将PC+4存入rd寄存器，并且跳转到（rs1）+imm地址
B-type	5	BEQ		5bit rs2		000		1100011	当（rs1）=（rs2)时，跳转至PC+imm
	6	BNE				001			当（rs1）！=（rs2)时，跳转至PC+imm
	7	BLT				100			当（rs1）<（rs2)时，跳转至PC+imm(有符号数比较)
	8	BGE				101			当（rs1）>=（rs2)时，跳转至PC+imm(有符号数比较)
	9	BLTU				110			当（rs1）<（rs2)时，跳转至PC+imm(无符号数比较)
	10	BGEU				111			当（rs1）>=（rs2)时，跳转至PC+imm(无符号数比较)
I-type	11	LB				000	5bit rd	0000011	读取（rs1）+imm处地址的8bit数据（SE），存入rd寄存器
	12	LH				001			读取（rs1）+imm处地址的16bit数据（SE），存入rd寄存器
	13	LW				010			读取（rs1）+imm处地址的32bit数据，存入rd寄存器
	14	LBU				100			读取（rs1）+imm处地址的8bit数据（0E），存入rd寄存器
	15	LHU				101			读取（rs1）+imm处地址的16bit数据（0E），存入rd寄存器
S-type	16	SB		5bit rs2		000		0100011	将rs2寄存器的低8bit数据，存入（rs1）+imm[取字对齐]处的内存
	17	SH				001			将rs2寄存器的低16bit数据，存入（rs1）+imm[取字对齐]处的内存
	18	SW				010			将rs2寄存器的32bit数据，存入（rs1）+imm[取字对齐]处的内存
I-type	19	ADDI				000	5bit rd	0010011	将（rs1）+imm存入rd寄存器 截图(Alt + A)
	20	SLTI				010			当（rs1）<imm时，将1存入rd寄存器（有符号数比较）
	21	SLTIU				011			当（rs1）<imm时，将1存入rd寄存器（无符号数比较）
	22	XORI				100			将（rs1）XOR imm的结果存入rd寄存器
	23	ORI				110			将（rs1）OR imm的结果存入rd寄存器
	24	ANDI				111			将（rs1）AND imm的结果存入rd寄存器
	25	SLLI	0000000			001			将（rs1）<<shamt的结果，存入rd寄存器
	26	SRLI				101			将（rs1）>>shamt的结果，存入rd寄存器
	27	SRAI	0100000			101			将（rs1）>>>shamt的结果，存入rd寄存器
R-type	28	ADD	0000000	5bit rs2		000	5bit rd	0110011	将（rs1）+（rs2）结果，存入rd寄存器
	29	SUB	0100000			000			将（rs1）-（rs2）结果，存入rd寄存器
	30	SLL	0000000			001			将（rs1）<<（rs2）[低5bit]结果，存入rd寄存器
	31	SLT				010			当（rs1）<（rs2）时，将1存入rd寄存器（有符号数比较）
	32	SLTU				011			当（rs1）<（rs2）时，将1存入rd寄存器（无符号数比较）
	33	XOR				100			将（rs1）XOR（rs2）结果，存入rd寄存器
	34	SRL				101			将（rs1）>>（rs2）[低5bit]结果，存入rd寄存器
	35	SRA	0100000			101			将（rs1）>>>（rs2）[低5bit]结果，存入rd寄存器
	36	OR	0000000			110			将（rs1）OR（rs2）结果，存入rd寄存器
	37	AND				111			将（rs1）AND（rs2）结果，存入rd寄存器

图 3: RISC-V 指令译码表

上图详细列出了 RISC-V 指令集的编码格式与功能。表格依据指令对操作数和立即数的不同使用方式，将指令集划分为 R、I、S、B、U、J 六种核心类型。通过分析可见，RISC-V 指令集具有高度的规整性：所有指令均为 32 位定长，且 opcode、rd、rs1 等关键字段在不同格式中的位置相对固定，通过上图，可以清晰地看出各个字段的含义和位置，更加准确地指导我完成代码逻辑。

二、 部分代码（与 MIPS 差异较大的部分）设计和实现

• Imm 的生成和代码实现

通过查阅相关资料和对 RISC-V 指令集的分析，我发现 RISC-V 指令集的指令格式和 MIPS 指令集的指令中的立即数生成方式是非常相似的。RISC-V 指令集也包含了 R 型、I 型、S 型、B 型、U 型和 J 型等指令格式。但是也有不同之处。RISC-V 中的立即数分类和产生方式如下图（见图 4）。

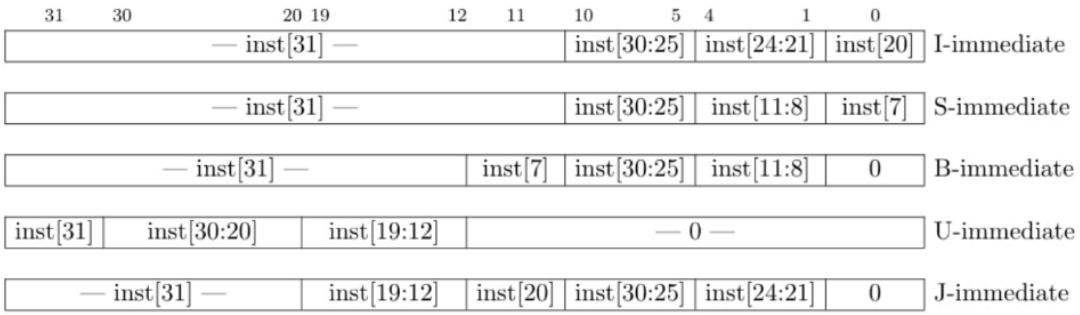

图 4: RISC-V 立即数表

我的 Imm 译码逻辑如下（见图 5）。

```
assign I_imm = {{20{instruction[31]}},instruction[31:20]};
assign S_imm = {{20{instruction[31]}},funct_length7,rd};
assign B_imm = {{21{instruction[31]}},instruction[7],instruction[30:25],instruction[11:8],1'b0};
assign U_imm = {{20{instruction[31]}},instruction[31:12],{12{1'b0}}};
assign J_imm = {{12{instruction[31]}},instruction[19:12],instruction[20],instruction[30:21],1'b0};
```

图 5: RISC-V 立即数生成代码

- **指令分类分析和代码实现**

相比于 MIPS 处理器，最明显的改变就是译码逻辑。在 RISC-V 处理器中，一共有 6 种不同的指令格式，分别是 R、I、S、B、U 和 J 类型，通过查询官方手册，我们可以清楚了解到不同指令的共性和差异（见图 6）。

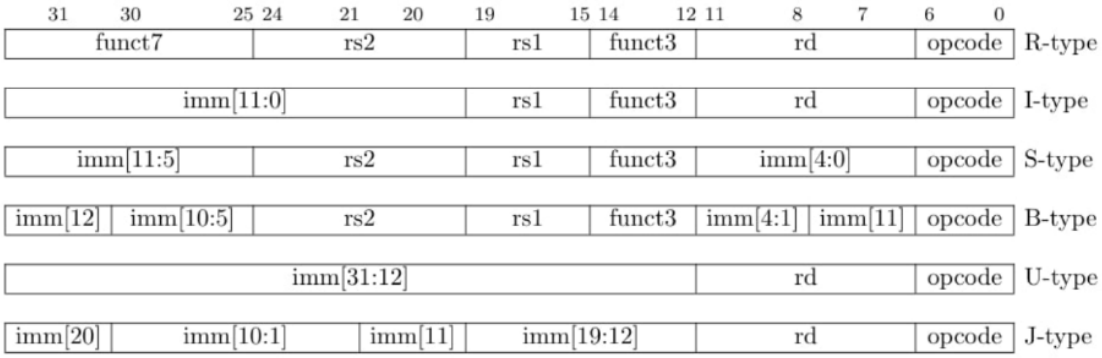

图 6: RISC-V 指令分类

我的具体代码，简单地利用了 Opcode 的值来判断指令类型，代码如下（见图 7）：

```
//指令码类型分类
wire U_type;
wire J_type;
wire I_type;
wire B_type;
wire S_type;
wire R_type;
wire I_LOAD_type;

assign U_type      = Opcode == 7'b0110111 | Opcode == 7'b0010111;
assign J_type      = Opcode == 7'b1101111;
assign I_type      = Opcode == 7'b0010011 | Opcode == 7'b1100111;
assign B_type      = Opcode == 7'b1100011;
assign S_type      = Opcode == 7'b0100011;
assign R_type      = Opcode == 7'b0110011;
assign I_LOAD_type = Opcode == 7'b0000011;
```

图 7: RISC-V 指令分类代码

- **PC 模块的设计和代码实现**

与 MIPS 多周期处理器相比，RISC-V 多周期处理器的 PC 模块设计和代码实现是相似的。但是也有很大不同之处。我在 Prj3 中设计的 MIPS 多周期处理器的 PC 模块设计和代码实现是会有两次跳转的，即：所有指令在 IW 阶段都会进行一次 PC+4 的计算，而跳转指令在 EX 阶段还会进行一次跳转。而在 RISC-V 多周期处理器中，PC 模块的设计和代码实现只有一次跳转，即：非跳转指令在 EX 阶段都会进行一次 PC+4 的计算，而跳转指令在 EX 阶段还会进行一次跳转。两者的代码实现对比如下图（见图 8、9）。

```
assign next_pc = Opcode == 7'b1101111 ? PC+J_imm :
                 Opcode == 7'b1100111 ? {tem_rdata1[31:1],1'b0} :
                 Opcode == 7'b1100011 ? PC+B_imm : 32'b0;
reg [31:0] reg_pc;
always @(posedge clk) begin
    if(rst) begin
        reg_pc <= 32'b0;
    end
    else if(current_stata == EX) begin
        if(is_jorb) begin
            reg_pc <= next_pc;
        end
        else begin
            reg_pc <= PC+4;
        end
    end
end
assign PC = reg_pc;
```

图 8: RISC-V 中 PC 模块

```
assign next_pc = branch    ? PC+imm_32_SE00       :
                 Opcode[1] ? {PC[31:28],tar,2'b0} :
                                          rsdata  ;
reg [31:0] reg_pc;
always @(posedge clk) begin
    if(rst) begin
       reg_pc <= 32'b0;
    end
    else if(current_stata == ID) begin
       reg_pc <= PC+4;
    end
    else if(current_stata == EX) begin
        if(is_jorb) begin
            reg_pc <= next_pc;
        end
    end
end
assign PC = reg_pc;
```

图 9: MIPS 中 PC 模块

三、 实验中的难点及实验收获

1 实验中的难点及 debug 过程

本实验中，我只花费了 2 个小时，查阅手册和相关资料完成译码表，花费 2 个小时完成代码部分。因为除了译码逻辑和一些细节之处，基本上可以说是与 Prj3 中 MIPS 处理器代码如出一辙。所以本实验完成代码过程中没有遇到什么难点。主要时间花费在 debug 上。

在 debug 过程中，我学到了一个之前没有注意到的知识点。当我在执行 shift 函数时，出现报错。我通过波形图发现我的 shifterop 非 1 即 0，是一个一位宽的信号。但是我需要它是一个两位宽的信号。波形图如下（见图 10）。

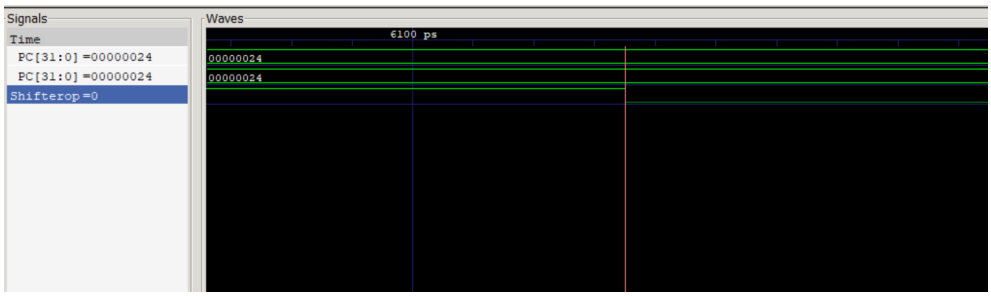

图 10: shifterop 出错的波形图

我返回代码检查后，发现原来我根本就没有 wire 这个信号。而且我编译通过了，它并没有报错。后面我学习相关知识后，了解到，这样的“未定义”并不会语法报错（虽然会有 warning），编译器会自动将它当作 0 处理。于是我在代码中添加了 wire shifterop，并且将其赋值为 2 位宽的信号（见图 11）。之后再运行 shift 函数时，发现运行成功。

```
wire [ 1:0] Shifterop;
assign Shifterop = funct_length7[5] == 1'b1 ? 2'b11 : funct_length3[2:1];
```

图 11: 添加 wire 后的代码

---

2 实验收获（两种指令集的对比）

通过对比我的周期计数器和指令计数器在同一个测试程序下的运行结果，我发现 RISC-V 多周期处理器的周期计数器和指令计数器的值相较于 MIPS 要有所优化，花费的周期数更少一些，虽然也不是特别明显。对比如下（见图 12、13）

```
101 tggetattr: Inappropriate ioctl for device
102 reset: before MMIO access...
103 reset: MMIO accessed
104 axi_firewall_unblock: firewall error status: 00000000
105 main: before DDR accessing...
106 main: DDR accessed...
107 reset: before MMIO access...
108 reset: MMIO accessed
109 [dinic] Dinic's maxflow algorithm: * Passed.
110 TOTAL CYCLES              : 1780688
111 TOTAL INSTRUCTIONS        : 19354
```

图 12: MIPS 指令集下运行结果

```
100 reset: before MMIO access...
101 reset: MMIO accessed
102 axi_firewall_unblock: firewall error status: 00000000
103 main: before DDR accessing...
104 main: DDR accessed...
105 reset: before MMIO access...
106 reset: MMIO accessed
107 [dinic] Dinic's maxflow algorithm: * Passed.
108 TOTAL CYCLES              : 1482774
109 TOTAL INSTRUCTIONS        : 16700
```

图 13: RISC-V 指令集下运行结果

通过上图，可以计算得出，在这个测试程序下，MIPS 多周期处理器的周期计数器为 1780688，而 RISC-V 多周期处理器的周期计数器为 1482774, 提升了约 16.8%。这说明 RISC-V 多周期处理器在指令集设计上进行了优化，减少了不必要的周期消耗。

查阅相关资料后，我了解到了这两类指令集的一些不同之处：RISC-V 的基础指令集更小，模块化的设计意味着设计者可以按需裁剪，从而可以用更少的逻辑门实现一个最小功能的处理器，这直接关系到面积、成本和功耗。除此之外，RISC-V 废除了分支延迟槽，使得设计现代高性能处理器（如多发射、乱序执行）的控制逻辑简单得多。MIPS 处理器为了兼容延迟槽，需要加入大量复杂的逻辑来处理和优化它，费力不讨好。

四、 思考题

本实验无思考题。

实验总结和感悟：

对 MIPS 与 RISC-V 的比较分析，可清晰观察到 RISC 指令集架构（ISA）的演进。MIPS 作为早期 RISC 架构的代表，其设计中包含的“分支延迟槽”等特性，虽在特定历史时期的简单流水线中有其价值，但对现代编译器优化和超标量处理器的设计构成了显著障碍。RISC-V 通过彻底移除此类设计遗留问题，实现了更为正交和规整的指令编码。其模块化的设计理念，允许实现按需裁剪的最小指令集，有效降低了硬件实现的复杂度与验证成本。因此，RISC-V 不仅是对 MIPS 的简单迭代，更是一场体系结构思想的革新。

五、 实验所耗时间

在课后，你花费了大约 30（代码完成 5 小时，Debug 完成 15 小时，完成实验报告 10 小时） 小时完成此次实验。

致谢：感谢蓝宇舟学长在我的指令译码表绘制上的帮助和朱徐塬助教在实验 debug 过程中给予的指导