中国科学院大学计算机组成原理实验课

实 验 报 告

学号：_2018K8009929030_　姓名：_热伊莱·图尔贡_　专业：_计算机科学与技术_

实验序号：__4__　实验名称：_RISCV 指令集处理器实现__

注 1：撰写此 Word 格式实验报告后以 PDF 格式保存在本地仓库的主目录下。文件命名规则：学号-prjN.pdf，其中学号中的字母“K”为大写，“-”为英文连字符，“prj”和后缀名“pdf”为小写，“N”为 1 至 4 的阿拉伯数字。例如：2019K8009929000-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：学号-prj5-projectname.pdf，例如：2019K8009929000-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 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（比如关键 RTL 代码段{包含注释}及其对应的逻辑电路结构、相应信号的仿真波形和信号变化的说明等）

1. 指令集

31 27	26 25	24 20	19 15	14 12	11 7	6 0	
funct7		rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]			rs1	funct3	rd	opcode	I-type
imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode	B-type
imm[31:12]					rd	opcode	U-type
imm[20|10:1|11|19:12]					rd	opcode	J-type

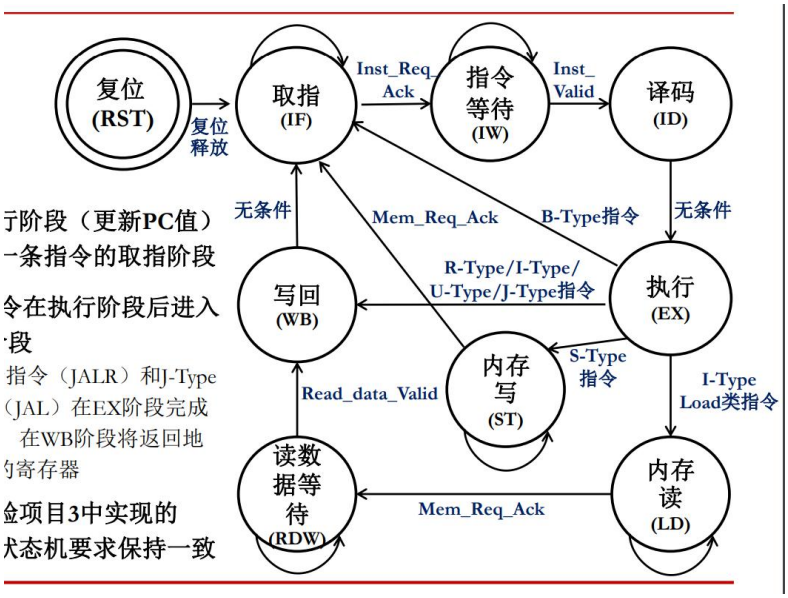
```
assign opcode = Instruction_r [6:0];
assign rd     = Instruction_r [11:7];
assign funct3 = Instruction_r [14:12];
assign rs1    = Instruction_r [19:15];
assign rs2    = Instruction_r [24:20];
assign funct7 = Instruction_r [31:25];
assign Address[31:0] = {alu_result [31:2],2'b0};
//type of operation
assign lui    = (opcode == 7'b0110111)?1:0;
assign auipc  = (opcode == 7'b0010111)?1:0;
assign jal    = (opcode == 7'b1101111)?1:0;
assign B_Type = (opcode == 7'b1100011)?1:0;
assign jalr   = (opcode == 7'b1100111)?1:0;
assign I_Type = (opcode == 7'b0010011)?1:0;
assign Store  = (opcode == 7'b0100011)?1:0;
assign Load   = (opcode == 7'b0000011)?1:0;
assign R_Type = (opcode == 7'b0110011)?1:0;
assign R_S    = R_Type && (funct3[0] && !funct3[1]);
assign R_C    = R_Type && !R_S;
assign U_Type = lui || auipc;
assign I_S    = I_Type && (funct3[0] && !funct3[1]);
assign I_C    = I_Type && !I_S;
assign Shift  = R_S || I_S;
```

扩展：

```
//extend
assign extend = jal?{{12{Instruction_r[31]}},Instruction_r[19:12],Instruction_r[20],Instruction_r[30:21],1'b0}:
                (jalr|I_C|Load)?{{20{Instruction_r[31]}},Instruction_r[31:20]}:
                B_Type?{{20{Instruction_r[31]}},Instruction_r[7],Instruction_r[30:25],Instruction_r[11:8],1'b0}:
                Store?{{20{Instruction_r[31]}},Instruction_r[31:25],Instruction_r[11:7]}:
                {Instruction_r [31:12],12'b0};//U_Type
```

2. 状态机描述：

使用“三段式”状态机描述



阶段（更新PC值）
一条指令的取指阶段
令在执行阶段后进入
段
指令（JALR）和J-Type
（JAL）在EX阶段完成
在WB阶段将返回地
寄存器
验项目3中实现的
状态机要求保持一致

“第一段”描述状态寄存器的同步状态跳转：

```
//describe the current_state
always @ (posedge clk) begin
        if (rst) begin
            current_state <= RST;
        end
        else begin
            current_state <= next_state;
        end
end
```

“第二段”根据状态机当前状态和输入信号，描述下一状态的计算逻辑：

```
//describe the next_state
always @ (*) begin
       case (current_state)
           RST:begin
              next_state = IF;
           end
           IF:begin
             if(Inst_Req_Ready)
                next_state = IW;
             else
                next_state = IF;
           end
           IW:begin
             if(Inst_Valid)
                next_state = ID;
             else
                next_state = IW;
           end
           ID:begin
                next_state = EX;
           end
           EX:begin
             if(B_Type)
                next_state = IF;
             else if(Store)
                next_state = ST;
             else if(Load)
                next_state = LD;
             else
                next_state = WB;
           end
           LD:begin
             if(Mem_Req_Ready)
                next_state = RDW;
             else
                next_state = LD;
           end
           ST:begin
             if(Mem_Req_Ready)
                next_state = IF;
             else
                next_state = ST;
           end
RDW:begin
  if(Read_data_Valid)
     next_state = WB;
  else
     next_state = RDW;
end
WB:begin
   next_state = IF;
end
default:
   next_state = current_state;
```

“第三段”根据状态机当前状态，描述不同输出寄存器的同步变化：

```
//describe synchronous changes in different registers
assign Inst_Ready     =((current_state == IW) || (current_state == RST))?1:0;
assign Inst_Req_Valid =(current_state == IF)?1:0;
assign Read_data_Ready=((current_state == RDW) || (current_state == RST))?1:0;
assign MemRead        =(current_state == LD)?1:0;
assign MemWrite       =(current_state == ST)?1:0;
```

3. PC，Instruction 和 Read_data 的更新

Read_data/Instruction/PC 三个值与单周期时不同，只有特定周期才更新，因此我分别设置了三个 reg 类型的变量用于这三个信号的更新：

```
//PC
always @(posedge clk) begin
       if(rst)
          PC_r <=32'b0;
       else if (current_state == EX)
          PC_r <=PC_next;
       else
          PC_r <= PC_r;
end
assign PC = (current_state == IF)?PC_r:PC;
//INSTRUCTION
always @(posedge clk) begin
       if (current_state == IW && (Inst_Valid && Inst_Ready))
          Instruction_r <= Instruction;
       else
          Instruction_r <= Instruction_r;
end

//Read_data
always @(posedge clk) begin
       if (current_state == RDW && (Read_data_Valid && Read_data_Ready))
          Read_data_r <= Read_data;
       else
          Read_data_r <= Read_data_r;
end
```

4. 其余设计与 MIPS 一致

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，仿真、云平台调试过程中的难点等）

1. 问题：忘记移位指令不在 alu 里，rdata 出错，导致 Address 出错
2. 问题：对 and，or，xor 等指令的 aluop 的选择出现了问题，导致 Write_data 和 pc 出现问题

解决方法：这两个问题的出现都不会直接显示出来，是在刘士祺助教的帮助下，通过看 benchmark 里的汇编文件往前追，才可以发现之前某次指令寄存器写入有问题。

三、 对讲义中思考题（如有）的理解和回答

四、 在课后，你花费了大约_36__小时完成此次实验。

五、 对于此次实验的心得、感受和建议（比如实验是否过于简单或复杂，是否缺少了某些你认为重要的信息或参考资料，对实验项目的建议，对提供帮助的同学的感谢，以及其他想与任课老师交流的内容等）

本次实验不是很复杂，所有信息和资料都在 ppt 和 RISC-V 手册上可以找到，但是由于没有寄存器的对照，一旦出现写入错误问题，就需要追到前 n 个指令，而且还需要一定的汇编基础。通过这次实验我对汇编语言有了一定的认识，非常感谢刘士祺助教的帮助。