浙江大学

本科实验报告

课程名称：计算机体系结构

姓　　名：刘思锐

学　　院：计算机科学与技术学院

系：

专　　业：计算机科学与技术

学　　号：3200102708

指导教师：陈文智

2022 年　11 月　30 日

浙江大学实验报告

课程名称：计算机体系结构 实验类型：综合

实验项目名称：Pipelined CPU supporting multi-cycle operations

学生姓名：刘思锐 专业：计算机科学与技术 学号：3200102708

同组学生姓名：陈镛屹 指导老师：陈文智

实验地点：宿舍 实验日期：2022 年 11 月 30 日

一、 实验目的和要求

1. Understand the principle of piplines that support multicycle operations.
2. Master the design methods of piplines that support multicycle operations.
3. Master verification methods of pipelined CPU supporting multicycle operations.

二、 实验内容和原理

1. 更新数据通路。相比于普通的五级流水线 CPU，IF、ID 阶段的功能不变，EXE 与 MEM 合二为一变成了 FU，在此阶段进行 ALU 运算、乘除法器、访问 mem 和跳转五项操作，WB 阶段通过多路选择器选择结果。
2. 完成所需的 ALU、乘法器、除法器、Mem 单元、Jump 单元。
3. 更新处理器控制模块，在 FU 阶段可执行多周期的操作，在执行多周期操作时 FU 之外的部分会被 stall。

三、 实验过程和数据记录

1. ALU

EN 信号为 1 并且 state 为 0 时，说明需要开始执行 ALU 操作；此时将控制信号置为输入的 ALUControl，A、B 置为输入 ALU A、ALU B，状态设为非 0 表示正在进行运算。状态 state 非 0 表明 ALU 执行中。操作结束后将 res 传出。

```
always@(posedge clk) begin
    if(EN & ~state) begin //state == 0
        Control <= ALUControl;
        A <= ALUA;
        B <= ALUB;
        state <= 1; // 只需要一个时钟
    end
    else state <= 0;
end
```

2. DIV

EN 信号为 1 并且 state、res_valid（其实只判断 state 应该就够了）均为 0 时，说明需要开始执行除法操作；A_reg、B_reg 置为输入 A、B，A_valid 和 B_valid 设为 1（这里单发射且运算时其他部分 stall，肯定不存在冲突），state 设为 1 表示正在进行运算。状态 state 非 0 表明除法器运算中。运算结束后恢复 state。

```
//两个源都valid才开始算，算完会自动设置res_valid
 always@(posedge clk) begin
    if(EN & ~state & ~res_valid) begin //state == 0
        A_reg <= A;
        A_valid <= 1;
        B_reg <= B;
        B_valid <= 1;
        state <= 1;
    end
    else if(res_valid)begin
        A_valid <= 0;
        B_valid <= 0;
        state <= 0;
    end
end
```

3. MUL

EN 信号为 1 并且 state 为 0 时，说明需要开始执行乘法操作；A_reg、B_reg 置为输入 A、B，A_valid 和 B_valid 设为 1（这里单发射且运算时其他部分 stall，肯定不存在冲突），state 的第 6 位设为 1。状态 state 非 0 表明乘法器运算中。

比较奇怪的是这里状态寄存器用的不是加法而是位移，并且乘法器似乎一直在计算，哪怕此时的输入输出没有意义，我们只是在一个特定的周期去除适当的值。

```
always@(posedge clk) begin
    if(EN & state == 0) begin //state == 0
        A_reg <= A;
        B_reg <= B;
        state[6] <= 1;
    end
    else state <= (state >> 1);
end

wire [63:0] mulres;
multiplier mul(.CLK(clk),.A(A_reg),.B(B_reg),.P(mulres));

assign res = mulres[31:0];
```

4. Mem

内存访问也是多周期，state 设置为 3 位（即[2:0]），和 MUL 一样用位移控制。EN 信号为 1 并且 state 为 0 时，将所有输入的传到内部并设置 state[3]。Mem 中数据的地址是由立即数 imm 与 rs1_data 相加后得到。在经过两个周期后 state[0]=1 返回结果。

```
reg[31:0] addr;
always@(posedge clk) begin
    if(EN & state == 0) begin //state == 0
        rs1_data_reg <= rs1_data;
        rs2_data_reg <= rs2_data;
        imm_reg <= imm;
        mem_w_reg <= mem_w;
        bhw_reg <= bhw;
        addr <= imm + rs1_data;
        state[1] <= 1;
    end
    else begin
        state <= (state >> 1);
    end
end
```

5. Jump

状态机方面与此前的组件基本同理，当需要启用时将输入进一步传到内部并设置 state。

```
always@(posedge clk) begin
    if(EN & ~state) begin //state == 0
        JALR_reg <= JALR;
        cmp_ctrl_reg <= cmp_ctrl;
        rs1_data_reg <= rs1_data;
        rs2_data_reg <= rs2_data;
        imm_reg <= imm;
        PC_reg <= PC;
        state <= 1;
    end
    else begin
        state <= 0;
    end
end
```

判断跳转是否发生时时可以复用 cmp_32 模块。跳转的地址 PC_jump 由输入生成，不跳转的地址 PC_wb 为 PC+4。

```
cmp_32 cmp(.a(rs1_data_reg), .b(rs2_data_reg), .ctrl(cmp_ctrl_reg), .c(cmp_res));
assign PC_jump = JALR ? (rs1_data_reg + imm_reg) : (PC_reg + imm_reg);
assign PC_wb = PC_reg + 32'd4;
```

6. RV32core

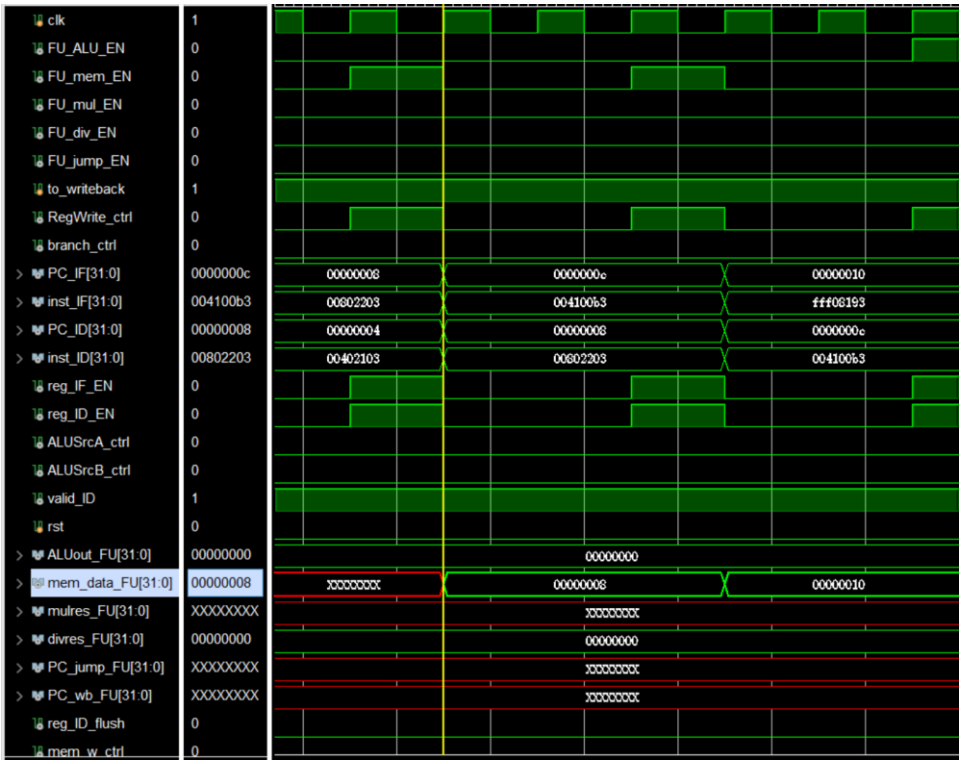
其中需要补全的部分为立即数生成器 imm_gen，ALU 两个数据输入的多路选择器 mux_imm_ALU_ID_A 和 mux_imm_ALU_ID_B 以及最后 WB 阶段的多路选择器 mux_DtR 的线路，按照原理图将这些器件连接妥当即可。

值得注意的是，第一，ALU_ID_A 和 ALU_ID_B 应该传入 ID 级的数据，不要错传成其他流水线部分的；第二，mux_DtR 的输入顺序与原理图上似乎并不严格对应，需要根据控制信号分析如何接线。

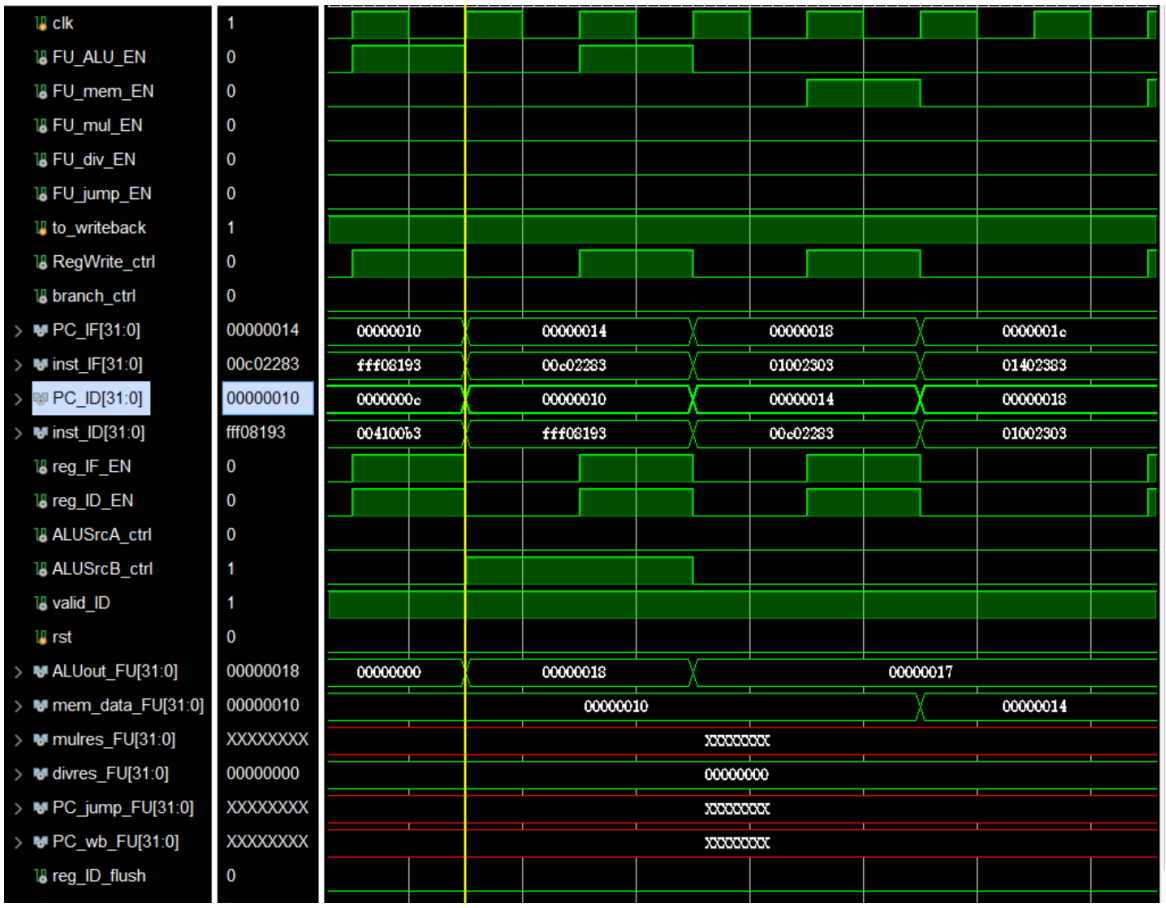
```
//to fill sth.in
 ImmGen imm_gen(.ImmSel(ImmSel_ctrl),.inst_field(inst_ID),.Imm_out(Imm_out_ID));

//to fill sth.in
MUX2T1_32 mux_imm_ALU_ID_A(.I0(rs1_data_ID),.I1(PC_ID),.s(ALUSrcA_ctrl),.o(ALUA_ID));

//to fill sth.in
MUX2T1_32 mux_imm_ALU_ID_B(.I0(rs2_data_ID),.I1(Imm_out_ID),.s(ALUSrcB_ctrl),.o(ALUB_ID));

MUX8T1_32 mux_DtR(.s(DatatoReg_ctrl),.I0(32'd0),.I1(ALUout_WB),.I2(mem_data_WB),.I3(mulres_WB),
.I4(divres_WB),.I5(PC_wb_WB),.I6(32'd0),.I7(32'd0),.o(wt_data_WB));        //to fill sth.in
```

四、 仿真结果分析

进行 load 指令，可见处理器正常停顿了两个时钟，在第三个时钟继续流水线，将正确的结果写回。

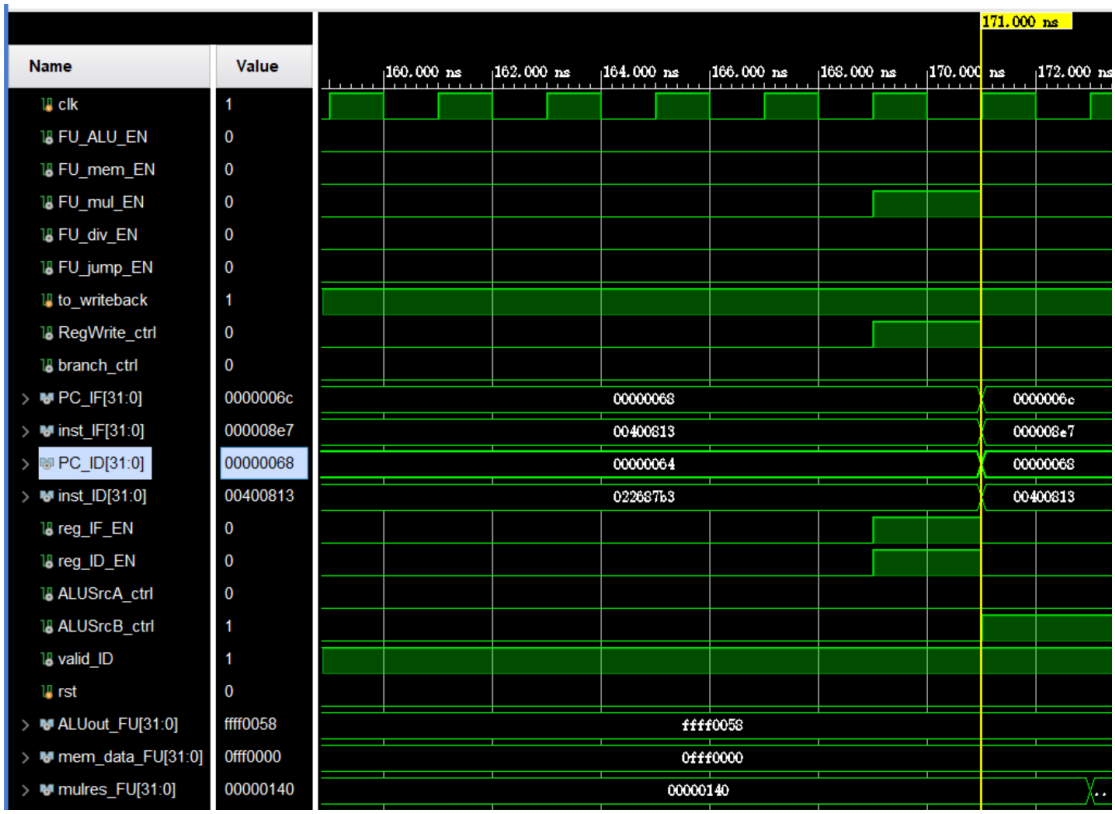


进行 ALU 运算时，首先 FU_ALU_EN 置为 1。执行过程需要一周期。执行结束后置为 0，RegWrite 信号置 1 将结果写入寄存器，此时 reg_IF_EN 和 reg_ID_EN 置 1，表示流水线继续执行。图中也可见运算结果正确。

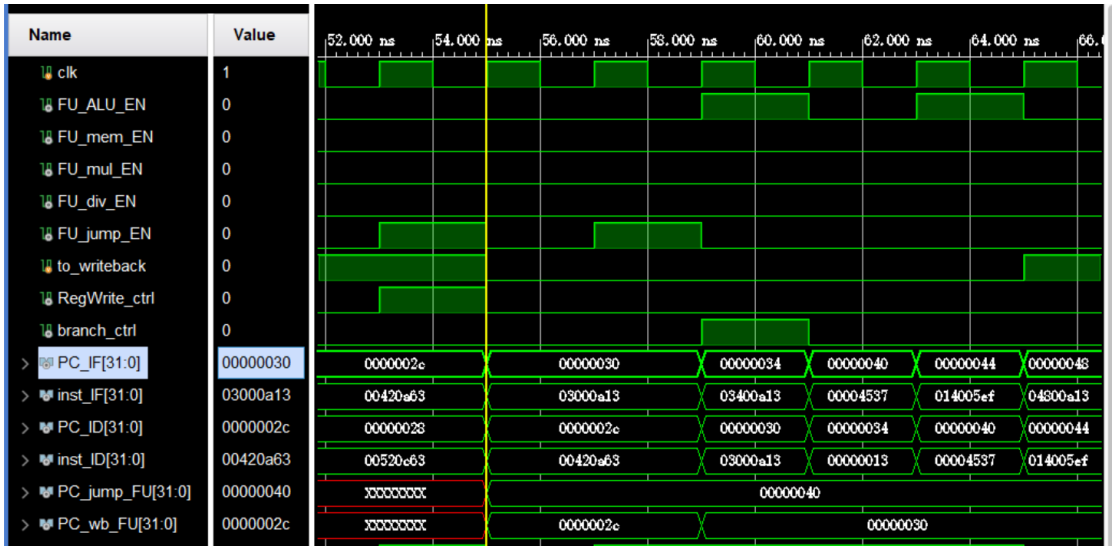


除法指令停顿拍数正常，经检查结果正确。

乘法指令同理。



进行跳转操作时，首先 FU_jump_EN 置 1，进行是否跳转的比较。发现需要跳转，则将branch_ctrl置 1，PC 选为跳转的 PC。当 jump 与 finish 均变为 1 时，跳转完成。



五、 讨论与心得

本次实验是实现多周期操作的流水线，是为动态流水线打下的基础与铺垫。相较于之前单周期的流水线，本流水线的结构更加紧凑，可以支持多周期指令的执行。在本实验中各指令执行时是停顿的，所以相对较容易理解。

在实验过程中，完成 RV32core 中的 mux_DtR 多选器时遇到了一些问题。

在 CtrlUnit 中，写回选择的信号是 use_FU，其信号顺序对应最终 WB 阶段多选器的选择。在最初实验时，我想当然将多选器从 0 开始排序，所以出现错误。

此外，对于 state 的使用，在乘法器中表示为其执行状态。在这里我们使用 state 向右移位的方式，该方式也可用加一或减一的方式进行替代，是一种很巧妙的计数方式