中国科学院大学计算机组成原理实验课

实 验 报 告

学号：2020K8009926006 姓名：游昆霖 专业：计算机科学与技术

实验序号：5.3 实验名称：增强功能型处理器设计

注 1：撰写此 Word 格式实验报告后以 PDF 格式保存在~/COD-Lab/reports 目录下。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。

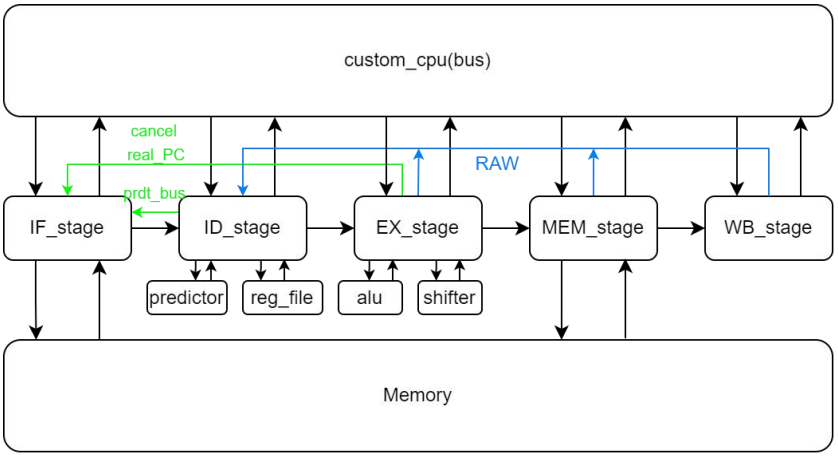
注 2：使用 git add 及 git commit 命令将实验报告 PDF 文件添加到本地仓库 master 分支，并通过 git push 推送到 GitLab 远程仓库 master 分支（具体命令详见实验报告）。

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图、相应信号的仿真波形和信号变化的说明等）

*说明：本次实验基于 RISCV 指令集实现，结构为五级流水线（IF、ID、EX、MEM、WB）,支持分支预测功能，由于每个模块的实现方式和 prj4 类似，以下仅列出数据流向，并结合波形对模块逻辑关键部分进行说明。

数据流向示意图：



上图中模块间数据传递均通过 custom_cpu 进行传递，为强调起见，将关于写后读（RAW）效应的数据通路使用蓝色在上图中进行表明，将关于分支预测的数据通路使用绿色在上图中进行表明。

1、 模块间数据传递

五级流水的每个模块均设计了相应代码，并用 custom_cpu.v 文件进行了封装。前一级给后一级的数据通过总线(AA_to_BB_bus)进行传递，且当 AA_to_BB_valid 和 BB_ready 信号同时为高时传输数据有效。

特别的，考虑到分支预测。将由 ID_stage.v 向 IF_stage.v 传递预测跳转地址和预测跳转结果 prdt_bus。由 EX_stage.v 向 IF_stage.v 传递预测失败信号 cancel 和 PC 正确结果 real_PC 以保证预测失败后流水线的正确执行。

考虑到单发射流水线中的写后读效应，分别从 EX、MEM、WB 阶段向 ID 阶段进行数据前馈，且优先级 EX>MEM>WB。考虑到例化 reg_file.v 文件时，同时连接读写端口，将 WB 写向寄存器的值通过 WB_to_RF_bus 前递至 ID 解读。

2、IF 阶段

该部分需要实现考虑分支预测时下一条指令地址 PC 的获取和已取得有效指令有效性的判断。以下为关键代码部分。

（1）状态机

```
always @(*) begin
                case (IF_cur_state)
                        `RST : begin
                                        IF_next_state = `IF;
```

```
                        end
                        `IF : begin
                                if(cancel)
                                        IF_next_state = `IF;
                                else if(Inst_Req_Valid & Inst_Req_Ready)
                                        IF_next_state = `IW;
                                else
                                        IF_next_state = `IF;
                        end
                        `IW : begin
                                if(cancel)
                                        IF_next_state = `IF;
                                else if(Inst_Ready & Inst_Valid)
                                        IF_next_state = `RDS;
                                else
                                        IF_next_state = `IW;
                        end
                        `RDS : begin
                                if(cancel)
                                        IF_next_state = `IF;
                                else if(ID_ready)
                                        IF_next_state = `SDD;
                                else
                                        IF_next_state = `RDS;
                        end
                        `SDD : begin
                                if(ID_ready)
                                        IF_next_state = `IF ;
                                else
                                        IF_next_state = `SDD;
                        end
                        default: begin
                                        IF_next_state = `RST;
                        end
                endcase
        end
```

IF 阶段的状态机共设置 RST、IF、IW、RDS、SDD 五个阶段，其中 RDS 阶段表示准备传输数据至 ID 阶段，SDD 表示传输数据完毕。

（2）控制信号

```
//Signal show whether this IF_pipeline really work
        always @(posedge clk) begin
```

```
                if(rst)
                        IF_work <= 1'b0;
                else begin
                        if(cancel)
                                IF_work <= 1'b0;
                        else if(IF_ready)
                                IF_work <= 1'b1;
                end
        end

//Control Unit
        assign IF_ready = ~IF_work | (IF_done & ID_ready);
        assign IF_done  = IF_cur_state ==`SDD & ~cancel;
```

由于需要进行分支预测，设置 IF_work 信号表示本条指令的取指是否有效。下一次 IF 阶段开始工作的条件是上一次分支预测失败，或上一个取指正确且被 ID 阶段接收。同时，为保证数据传输有效性，设置本阶段完成信号。

（3）PC 更新逻辑

```
always @(posedge clk) begin
                if(rst)
                        PC <= 32'b0;
                else if(cancel)  //CHECK the COND
                        PC <= real_PC;
                else if(IF_done & ID_ready) begin
                        PC <= prdt_go ? prdt_tar : PC +4;
                end
        end
```

IF 阶段在 RDS 状态将本条指令的 PC 传向 ID 阶段，从而在 SDD 阶段按照 ID 阶段前馈的分支预测信号进行更新。当需要进行分支预测的指令流转至 EX 阶段时，将得到该预测是否准确，并前递 cancel 信号和正确的 PC。若分支预测失败，则 PC 需要再次进行更新以维护流水线的正确执行。

仿真波形说明：

上图中 IF 模块在 RDS 阶段（08）向 ID 模块传递信息并拉高有效信号，在 SDD 阶段（10）获得 ID 模块前馈的信息，由于预测进行跳转，在 SDD 阶段进行了 PC 的跳转更新，在下一个 IF 阶段（02）接受到 EX 阶段前馈的 cancel 信号，即分支预测失败，此时暂时拉低本模块操作有效信号 IF_work，拉低 Inst_Req_Valid，设置次态仍为 IF，根据前馈信息将 PC 更新为正确值，再进行取指及恢复 IF_work。

（4）数据传递

```
assign IF_to_ID_valid = IF_cur_state ==`RDS & ID_ready & ~cancel & IF_work;
assign IF_to_ID_bus ={
                      PC              ,       //63:32
                      Instruction_Reg         //31:0
                      };
```

IF 阶段需要向 ID 阶段传输的数据及控制信号如上所示。特别的，设置 IF_to_ID_valid 信号在 RDS 阶段拉高，从而可在 SDD 阶段得到 ID 前馈的是否跳转和跳转结果，如果分支预测失败，可以在下一个 IF 阶段得到 EX 前馈的 cancel 信号和真实 PC。

3、ID 阶段

该部分需要实现指令译码，分支预测，和寄存器堆 Reg_file 的交互，写后读的处理，以及数据传递。

（1）指令译码

和 prj4 所实现基本一致，对 IF 阶段传输过来的指令进行译码即可。

（2）分支预测

predictor.v

```
`include "turbo_macro.v"

module predictor(
        input   clk,
        input   rst,

        input   Branch, //type is branch or jump
        input   prdt_work,//limit the state shift to one clk
        //the signal is equal to cancel hold
        //signal of not taken
        input   cancel,
        output  prdt_br
);

        reg [4:0] prdt_cur_state;
        reg [4:0] prdt_next_state;

        always @ (posedge clk) begin
                if(rst)
                        prdt_cur_state <= `RST ;
                else
                        prdt_cur_state <= prdt_next_state ;
        end

        //note that state will not change unless Branch
        always @ (*) begin
                case (prdt_cur_state)
                        `RST : begin
                                if(rst)
                                        prdt_next_state = `RST ;
                                else
                                        prdt_next_state = `S_Taken ;
                        end
                        `S_Taken : begin
                                if(prdt_work & cancel & Branch)
                                        prdt_next_state = `W_Taken ;
                                else
                                        prdt_next_state = `S_Taken ;
                        end
                        `W_Taken : begin
```

```
                                if(prdt_work & cancel & Branch)
                                        prdt_next_state = `W_NTaken ;
                                else if (prdt_work & ~cancel & Branch)
                                        prdt_next_state = `S_Taken ;
                                else
                                        prdt_next_state = `W_Taken ;
                        end
                        `W_NTaken : begin
                                if(prdt_work & cancel & Branch)
                                        prdt_next_state = `S_NTaken ;
                                else if (prdt_work & ~cancel & Branch)
                                        prdt_next_state = `W_Taken ;
                                else
                                        prdt_next_state = `W_NTaken ;
                        end
                        `S_NTaken : begin
                                if(prdt_work & cancel & Branch)
                                        prdt_next_state = `S_NTaken ;
                                else if(prdt_work & ~cancel & Branch)
                                        prdt_next_state = `W_NTaken ;
                                else
                                        prdt_next_state = `S_NTaken ;
                        end
                        default:
                                prdt_next_state = `RST;
                endcase
        end

        assign  prdt_br = (prdt_cur_state == `S_Taken) | (prdt_cur_state == `W_Taken);
```

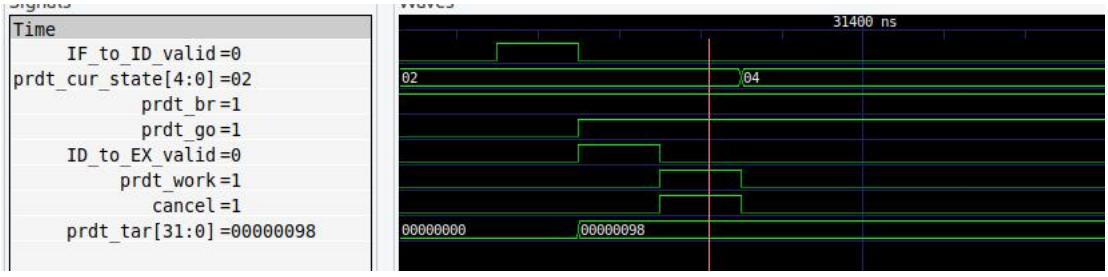
Custom_cpu.v 中相关通路

```
//predict Unit
        assign prdt_go  = (I_jalr | J_type) | (B_type & prdt_br);
        assign jalr_tar = rs1_value + I_imm;
        assign prdt_tar =       B_type ? ID_PC + B_imm :
                                J_type ? ID_PC + J_imm :
                                I_jalr ? {jalr_tar[31:1] ,1'b0} :
                                0; //default
        assign prdt_bus = {prdt_go , prdt_tar};
        always @(posedge clk) begin
                if(ID_to_EX_valid)
                        prdt_work <= 1'b1;
                else
                        prdt_work <= 1'b0;
```

```
        end


        predictor prdt_inst (
                .clk            (clk),
                .rst            (rst),
                .Branch         (B_type),
                .prdt_work      (prdt_work),
                .cancel         (cancel),
                .prdt_br        (prdt_br)
        );
```

为提高分支预测精度，使用 2 位分支预测。为控制 1 次分支预测只产生一次分支预测状态转移，限制 cancel 只维持一拍(EX 阶段完成)，且设置 prdt_work 表示 cancel 有效的周期。

仿真波形说明：



Prdt_br 根据此时预测器的状态确定，当预测器处于强接受状态（02）或弱接受状态（04）时该信号拉高。当 ID 译码得到无条件跳转指令，或条件跳转指令且 prdt_br 为高时拉高 prdt_go 信号，表示分支预测需要进行跳转。由于 EX 可在一个阶段内给出运算结果，在 ID_to_EX_valid 拉高的下一个周期拉高 prdt_work 信号，表示 cancel 信号实际作用，并令预测器根据 cancel 信号进行状态转移。如上图，由于 cancel 信号在对应周期拉高，令预测器从强接受状态转移至弱接受状态。

（3）与寄存器堆 Reg_file 的交互

```
//Channel to regfile
        //read data from ID, write data from WB
        assign RF_raddr1 = rs1;
```

```
        assign RF_raddr2 = rs2;
        assign {WB_RF_wen , WB_RF_waddr , WB_RF_wdata} = WB_to_RF_bus;

        reg_file reg_file_inst(
                .clk(clk),
                .raddr1(RF_raddr1),
                .raddr2(RF_raddr2),
                .rdata1(RF_rdata1),
                .rdata2(RF_rdata2),
                .waddr(WB_RF_waddr),
                .wdata(WB_RF_wdata),
                .wen(WB_RF_wen)
        );
```

由于读写寄存器堆均在 ID 阶段进行，因此将隐退指令 WB 阶段的写寄存器信号传至 ID 阶段，作为寄存器堆写端口数据，将本条指令译码所得地址作为寄存器堆读地址。

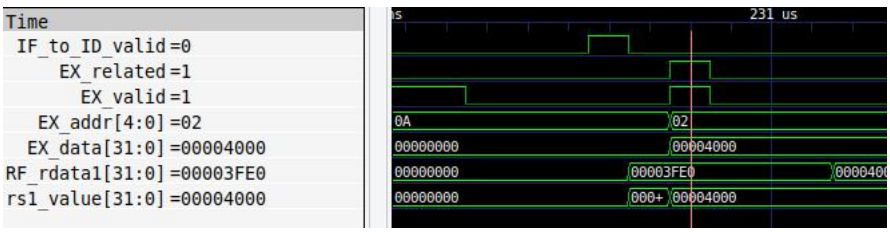
（4）写后读的处理

```
//register value considering pipeline
        assign {EX_load , EX_valid , EX_addr , EX_data} = EX_fw_bus;
        assign {MEM_load , MEM_done , MEM_valid , MEM_addr , MEM_data} = MEM_fw_bus;
        assign {WB_valid , WB_addr , WB_data} = WB_fw_bus;

        assign rs1_value =  ~(|rs1)                   ? 32'b0         :
                        EX_valid  & (EX_addr  == rs1)   ? EX_data       :
                        MEM_valid & (MEM_addr == rs1)   ? MEM_data      :
                        WB_valid  & (WB_addr  == rs1)   ? WB_data       :
                        RF_rdata1;
        assign rs2_value =  ~(|rs2)                   ? 32'b0         :
                        EX_valid  & (EX_addr  == rs2)   ? EX_data       :
                        MEM_valid & (MEM_addr == rs2)   ? MEM_data      :
                        WB_valid  & (WB_addr  == rs2)   ? WB_data       :
                        RF_rdata2;

//Control Unit
        assign EX_related = (|EX_addr) & EX_valid & (rs1==EX_addr | rs2==EX_addr);
        assign MEM_related = (|MEM_addr) & MEM_valid & (rs1==MEM_addr | rs2==MEM_addr);

        assign block    = ( EX_load & EX_related ) | (MEM_load & MEM_related & ~MEM_done);
        assign ID_done  = ~block & ~cancel;
       always @(posedge clk) begin
```

```
            if(rst)
                    ID_work <= 1'b0;
            else begin
                    if(cancel)
                            ID_work <= 1'b0;
                    else if(ID_ready)
                            ID_work <= IF_to_ID_valid;
            end
    end

    assign ID_ready = ~ID_work | (ID_done & EX_ready);
    assign ID_to_EX_valid = ID_work & ID_done & EX_ready;
```

考虑写后读效应，从寄存器堆对应地址取出来的值可能会被之前指令的后续阶段改变，因此需要由 EX、MEM 和 WB 阶段进行数据前递。通过地址和有效信号判断前递数据是否和寄存器读数据相关，并选择得到该寄存器读地址对应的正确数据，选择优先级为 EX>MEM>WB，与本条指令约接近则优先级越高。特别的，当 EX 阶段或 MEM 阶段的指令为 load 类型且数据相关时，需要进行阻塞，等待该条指令执行完访存操作后才可得到正确数据，从而继续运行流水线。

仿真波形说明：



光标所示位置，由于 EX 传输的信号有效时，其写地址和寄存器堆的读地址相同，因此考虑写后读效应，使用 EX 传递的写数据而非直接从寄存器堆取出的数据表示 rs1 号寄存器的数据。

（5）数据传递

```
//to EX bus
        //send rs2_value to get Write_data later
        assign R_Icalc  = R_type | I_calc;
        assign load     = I_load;
```

```
    assign store    = S_type;
    assign jump     = I_jalr | J_type;
    assign RF_wen   = R_type | I_type | J_type | U_type;
    assign RF_waddr = rd;
    assign RF_ID_alter = (I_jalr | J_type) | U_lui | U_auipc ;
    assign RF_ID_alter_data =      I_jalr | J_type ? ID_PC +4 :
                                   U_lui           ? U_imm         :
                                   U_auipc         ? ID_PC + U_imm :
                                   0; //default
    assign ID_to_EX_bus = {
                              ID_PC,               //247:216
                              prdt_go,             //215:215
                              prdt_tar,            //214:183
                              ALU_op,              //182:182
                              ALU_A,               //181:150
                              ALU_B,               //149:118
                              Shifter_op,          //117:116
                              Shifter_A,           //115:84
                              Shifter_B,           //83:79
                              B_type,              //78:78
                              R_Icalc,             //77:77
                              load,                //76:76
                              store,               //75:75
                              jump,                //74:74
                              funct3,              //73:71
                              RF_wen,              //70:70
                              RF_waddr,            //69:65
                              RF_ID_alter,         //64:64
                              RF_ID_alter_data,    //63:32
                              rs2_value            //31:0
                              };
```

ALU、Shifter 操作数和控制信号与多周期处理器类似。为节约传输数据的位宽，根据后续阶段的操作类型对已有指令进行简单分类，且将 ID 阶段可确定的寄存器堆写数据进行选择后再传输。

4、EX 阶段

该阶段需要与 ALU、Shifter 进行交互，判断分支预测是否成功并给出真实 PC，前递本条指令写寄存器堆的相关信息，向后传递访存相关数据及当前确定

的写寄存器堆相关数据。

（1）ALU、Shifter 的交互

将 ID 阶段传递的控制信号和操作数传递至对应组件即可，与多周期类似。

（2）判断分支预测并进行相应操作

```
//Test predict result , cancel if fail
        assign br_en = B_type & ( ( ~funct3[2] & ~funct3[0] &  ALU_Zero )       |
                                  ( ~funct3[2] &  funct3[0] & ~ALU_Zero)        |
                                  (  funct3[2] & ~funct3[0] &  ALU_Result[0])   |
                                  (  funct3[2] &  funct3[0] & ~ALU_Result[0])
                                );
        assign real_go = br_en | jump;

        always @(posedge clk) begin
                if(ID_to_EX_valid)
                        cancel_hold <= 1'b1;
                else
                        cancel_hold <= 1'b0;
        end
        assign cancel = ( real_go ^ prdt_go ) & cancel_hold;    //set high if not equal
        assign real_PC = real_go ? prdt_tar : EX_PC +4;
```

根据 ALU 所计算的结果获取分支预测是否成功，将真实跳转结果与预测结果进行异或及得到了 cancel 信号，cancel 为高表示分支预测失败需要重启流水线。同时设置 cancel_hold 信号，保证 cancel 只拉高一拍，从而保证流水线和分支预测器的正确操作。

仿真波形说明：

此处根据 ALU 计算的结果获取了真实的跳转结果 real_go，和预测结果 prdt_go 不同，即分支预测失败，应当拉高 cancel。设置 cancel_hold 延后

ID_to_EX_valid 一拍拉高，从而使得 cancel 只拉高一拍。

（3）前递数据

考虑写后读效应，将本条指令是否写寄存器、写地址及 EX 当前所得数据向 ID 阶段前递。同时传递本条指令是否为 load 类型，如是且数据相关，下一条指令的 ID 阶段将阻塞至本条指令流向 WB 并访存获得数据。

（4）向后传递

```
always @(posedge clk) begin
                if(rst)
                        EX_work <= 1'b0;
                else begin
                        if(EX_ready)
                                EX_work <= ID_to_EX_valid;
                end
        end

        assign EX_done = 1'b1; //always done in one cycle
        assign EX_ready = ~EX_work | (EX_done & MEM_ready);
        assign EX_to_MEM_valid = EX_work & EX_done & MEM_ready;
//send MEM bus
        assign EX_to_MEM_bus = {
                                EX_PC,                  //145:114
                                load,                   //113:113
                                store,                  //112:112
                                Address,                //111:80
                                load_tag,               //79:78
                                funct3,                 //77:75
                                Write_strb,             //74:71
                                Write_data,             //70:39
                                RF_wen,                 //38:38
                                RF_waddr,               //37:33
                                RF_EX_alter,            //32:32
                                RF_EX_alter_data        //31:0
                                };
```

向后传递写寄存器堆、访存和隐退指令的相关信号。

5、MEM 阶段

该阶段需要实现访存及相关数据的处理，前递写寄存器堆相关数据，向后传递最终写寄存器堆相关信号和隐退指令信号。

（1）访存状态机

```
always @(*) begin
                case(MEM_cur_state)
                        `RST: begin
                                MEM_next_state = `BSL;
                        end

                        `BSL: begin
                                if(MEM_work) begin
                                        if(load | store)
                                                MEM_next_state = `SL;
                                        else
                                                MEM_next_state = `SLD;
                                end
                                else
                                        MEM_next_state = `BSL;
                        end

                        `SL: begin
                                if(MEM_work) begin
                                        if(load & Mem_Req_Ready)
                                                MEM_next_state = `RDW;
                                        else if(store & Mem_Req_Ready )
                                                MEM_next_state = `SLD;
                                        else
                                                MEM_next_state = `SL;
                                end
                                else
                                        MEM_next_state = `SL;
                        end

                        `RDW: begin
                                if(Read_data_Ready & Read_data_Valid)
                                        MEM_next_state = `SLD;
                                else
                                        MEM_next_state = `RDW;
                        end

                        `SLD: begin
```

```
                                if(EX_to_MEM_valid) //WB ready
                                        MEM_next_state = `BSL;
                                else
                                        MEM_next_state = `SLD;
                        end

                        default: begin
                                MEM_next_state = `RST;
                        end
                endcase
        end
```

整体流程和多周期访存部分的状态转移类似，额外考虑了模块传输数据的有效信号。对于访存读写的数据，操作基本和多周期处理器类似。

（2）前递写寄存器堆信号

与 EX 阶段类似，但本阶段需要额外添加 MEM_done，表示访存是否完成，所取得的相关数据是否有效。

（3）向后传递数据

```
always @(posedge clk) begin
        if(rst)
                MEM_work <= 1'b0;
        else begin
                if(MEM_ready)
                        MEM_work <= EX_to_MEM_valid;
        end
end

always @(posedge clk) begin
        if(rst)
                MEM_to_WB_valid_hold <= 1'b0;
        else begin
                if(EX_to_MEM_valid)
                        MEM_to_WB_valid_hold <= 1'b1;
                else if(MEM_to_WB_valid)
                        MEM_to_WB_valid_hold <= 1'b0;
        end
end

assign MEM_done = MEM_cur_state == `SLD ;
```

```
assign MEM_ready = ~MEM_work | (MEM_done & WB_ready);
assign MEM_to_WB_valid = MEM_work & MEM_done & WB_ready & MEM_to_WB_valid_hold;
```

MEM_work 表示本次 MEM 访存有效，MEM_done 表示访存结束，WB_ready 表示 WB 阶段数据准入，且同时设置 MEM_to_WB_valid_hold 保证 MEM_to_WB_valid 只能拉高一拍。以上信号共同作用，保证了后续隐退信号最高位写使能信号只拉高一拍。

仿真波形说明：

当 EX_to_MEM_valid 拉高时，将 MEM_to_WB_valid_hold 拉高，当 MEM_to_WB_valid 拉高是，将其拉低，从而保证 MEM 不会重复将相同的隐退指令传递至 WB 阶段。特别的，当 EX_to_MEM_valid 和 MEM_to_WB_valid 同时为高的时候，表示 EX 已经向 MEM 传递了新的数据，不需担心重复隐退的问题，此时为使前一条指令成功隐退，仍需保持 MEM_to_WB_valid_hold 为高。

6、WB 阶段

该阶段需要实现隐退指令相关信号的处理、向 ID 前馈寄存器堆写信号。

（1）隐退指令相关信号的处理

```
always @(posedge clk) begin
        if(rst)
                WB_work <= 1'b0;
        else begin
                if(WB_ready)
                        WB_work <= MEM_to_WB_valid;
        end
end
```

```
assign WB_done = 1'b1;  //always done in one clk
assign WB_ready = ~WB_work | WB_done;
//inst retire
        //hold inst_retire_valid to one cycle to count valid inst
        //hold inst_retire_wen to one cycle to avoid repeated comparison
assign inst_retire_valid = WB_done & WB_work ;
assign inst_retire_wen = inst_retire_valid & WB_RF_wen ;
assign inst_retire = {inst_retire_wen , RF_waddr , RF_wdata,WB_PC};
```

inst_retire_valid 表示该条指令成功隐退，inst_retire_wen 则表示需要写寄存器的指令成功隐退，用于进行比对。

（2）前馈信号的处理

和 EX 阶段类似，且为最终写入寄存器堆的数据。

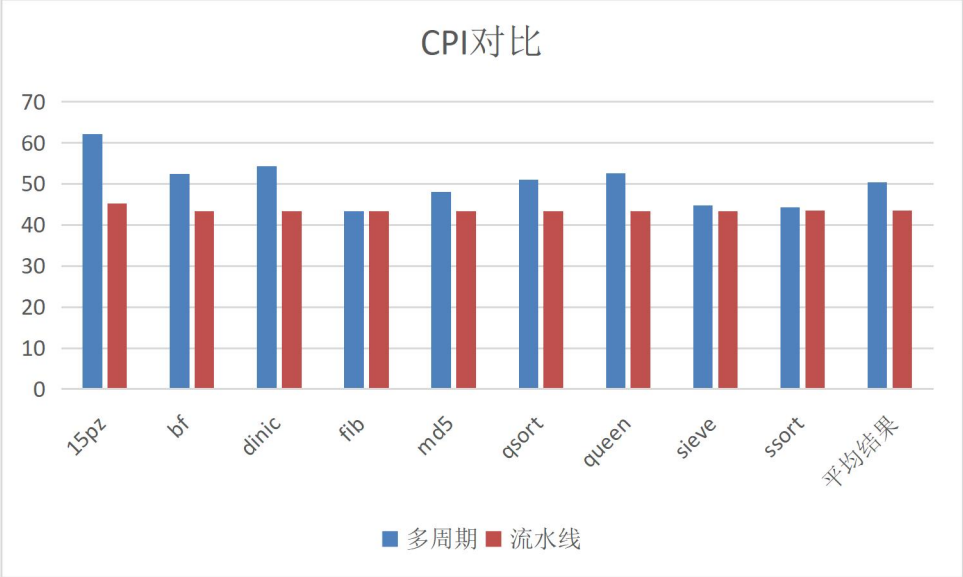
7、性能计数器

与多周期处理器计数对象相同，分散在各模块中，并基于各个模块的对应条件进行略微改动。

RISCV 指令集下多周期处理器和五级流水线处理器（未加 cache）性能比对如下：

	处理器	时钟周期数	指令总数	访存延迟周期数	CPI
15pz	多周期	324238832	5224477	90004486	62.06
	流水线	236498270	5224477	94278671	45.27
bf	多周期	23775610	452850	3956925	52.50
	流水线	19655008	452850	3947242	43.40
dinic	多周期	906059	16687	171982	54.30
	流水线	723066	16687	172866	43.33
fib	多周期	110676382	2549521	183886	43.41
	流水线	110479201	2549521	185795	43.33
md5	多周期	236203	4911	22927	48.10
	流水线	212774	4911	22781	43.33
qsort	多周期	483070	9476	65405	50.98
	流水线	410526	9476	66011	43.32
queen	多周期	4284106	81486	637557	52.57
	流水线	3532577	81486	640402	43.35
sieve	多周期	456038	10191	14335	44.75

	流水线	441542	10191	14452	43.33
ssort	多周期	27431823	619041	514622	44.31
	流水线	26922713	619041	554778	43.49
平均结果	多周期	54720902.56	996515.56	10619125	50.33
	流水线	44319519.67	996515.56	11098110.89	43.57



由上图可见，经过流水线对组件的复用，使不同指令可以进行并行计算，整体时钟周期数和 CPI 均有了一定的降低，但降低幅度仍未符合流水线的预期。进一步对访存周期进行统计对比，可见访存延迟仍占用了大量时间周期，降低了指令并行度，阻碍了流水线运行效率的提高，需要在后续实验中加入高速缓存 cache 进行进一步优化。

7、性能优化

由于在性能评测方法发布时，流水线和 cache 均已经实现，因此将二者作为整体进行代码优化，且由于将时钟频率修改为 300hz 时，未加 cache 的流水线 bit-gen 任务中 WNS 均大于 0.1，而在加上 cache 后有部分 bit-gen 任务出现 WNS 小于 0 的情形，且查看报告可见关键路径与流水线处理器的架构无关，出现在 cache 部分，因此主要对后续 cache 进行结构优化的考虑，详见 cache 部

分报告。

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）

问题 1：隐退指令比对问题

根据仿真报错发现多条指令重复与第一条写寄存器堆隐退指令进行比较导致出错。询问芦溶民助教本次实验中比对机制以及查看波形，发现 WB 直接使用寄存器堆写使能信号作为隐退指令最高位，拉高了多个周期造成隐退异常。

设置了 inst_retire_valid 和 inst_retire_wen 等信号，保证了隐退指令最高位只拉高一拍，从而正确进行比对。

问题 2：分支预测与 PC 更新逻辑

PC 更新出现分支预测指令之前额外更新、分支预测失败后恢复错误、和指令对应错误等问题，通过反复查看波形，进行了多处更改。

首先，以倒数第二个状态 RDS 作为 IF 向 ID 传输指令有效信号，从而在最后一个状态 SDD 可以得到 ID 阶段前馈的分支预测结果和地址，并在此状态进行分支预测更新或顺序更新。

其次，根据以上状态设置，可以在下一次流程的 IF 状态获得 EX 阶段前馈的分支预测是否成功信号 cancel 以及真实 PC，如果分支预测失败，则需设置次态仍为 IF，并重新进行取指。

另外，为保证指令内存响应正确，Inst_Req_Valid 同样需要考虑 cancel 指令，被取消的 PC 不进行取指，防止 PC 和 Instruction 不对应的问题。

问题 3：IF 阶段堵塞、预测状态转移异常

查看波形过程中，发现当分支预测失败时，IF 阶段会产生阻塞，且 2 位分支预测将直接从强接受状态转移至强拒绝状态，不符合预期。

设置 cancel_hold 信号，控制 cancel 信号只维持一拍，同时对分支预测器添加相关通路，保证仅在分支预测时改变，且一次分支预测只进行一次状态转移，从而进一步提高分支预测精度，增加流水线效率。

问题 4：IF 阶段和 MEM 阶段的状态转移

查看波形中发现数据传递出错，出现遗漏或重复的现象。

设置 IF 状态机末态转初态条件为 ID_ready，因为此时该条指令可被向后传递，防止取得的指令由于 ID 阶段的阻塞导致遗漏。且此时状态机才满足 PC 更新条件。

设置 MEM 状态机末态转初态条件为 EX_to_MEM_valid，这是为了保证 MEM 的 PC 更新，使得 WB 阶段隐退的 PC 不会重复写。且由于 WB 阶段操作只需 1 周期即可完成，WB_ready 始终为高，所以不需考虑数据遗漏问题。

问题 5：内存旁路访问请求冲突问题

medium 组 max 和 select-sort 任务在本地仿真和云平台仿真均无法结束，直至运行超时。但所有任务均可通过 fpga_eval 及硬件仿真，且在后续添加 icache 和 dcache 后这两个任务的仿真也可正常通过。

查看框架代码 cpu_test_top.v 可知 CPU 的 inst_req_valid 和 Memread

将通过 inst_if_wrapper 和 mem_if_wrapper 转换为 cpu_inst_arvalid 和 cpu_mem_arvalid。而在 cpu_to_mem_axi_2x1_arb.v 中二者是并列条件，只有一个有效。因此取指和访存不能同时进行，将 MemRead 信号传输至 IF 模块，在指令请求时进行考虑。

三、 对讲义中思考题（如有）的理解和回答

本次实验无思考题。

四、 在课后，你花费了大约____30____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）

心得感受:

相比其余选做实验，本实验难度较大，需要对五级流水不同模块的功能和模块之间的数据传递有较为深刻的理解。分支预测的应用和写后读效应的存在使得本实验对信号传递的时序精确度提出了更高的要求，以保证 PC 能够正确更新、寄存器堆数据能够正确获取、流水线能够正确而高效的工作和重启。另外，在进行基于 RISCV 指令集实现的多周期处理器和流水线处理器性能对比时，发现由于访存延迟过高，流水线频繁阻塞，性能和多周期相差不大，需要后续实验添加高速缓存 cache 以进一步提高处理器性能。

致谢:

感谢常轶松老师和陈欲晓在硬件仿真工具方面提供的帮助，感谢芦溶民助教对隐退指令比对机制的讲解，感谢刘士祺助教对模块间接口规范的介绍。