Lab 5 报告

学号：2020K8009929034、2020K8009929019、2020K8009929037
姓名：胡康、李子恒、吕星宇
箱子号：05

一、实验任务（10%）

在 lab4 的基础上，添加乘除法指令，并完成可以配套操作的访存指令（ld 和 st）。此外，完成更多的转移指令，使 CPU 的指令增加到 46 条指令。

二、实验设计（40%）

（一）总体设计思路

1. 在译码阶段，复用 decoder 完成指令的相应译码，并复用部分指令的数据通路和控制通路。

2. 对于乘除法，调用 IP 核来完成相应的操作。

3. 新增的四条跳转指令功能与已有的 beq、bne 非常相似，只是跳转的判断条件不同，因此设计上整体复用原有的数据通路，在其基础上增加对应的判断条件。

4. 新增的六条访存指令整体数据通路亦可复用原有的 ld.w 和 st.w 指令，并在此基础上修改数据的选择和加载信号，从而产生正确结果。

（二）重要模块 1 设计：除法器（以无符号为例）

定制 IP 核完成无符号除法，我们的工作主要是完成握手信号的连接。

1、工作原理

```
signed_divider my_signed_divider(
    .aclk                   (clk),
    // S_AXIS_DIVISOR: rk
    .s_axis_divisor_tdata   (es_alu_src2),
    .s_axis_divisor_tready  (divisor_ready),
    .s_axis_divisor_tvalid  (div_valid),
    // S_AXIS_DIVIDEND: rj
    .s_axis_dividend_tdata  (es_alu_src1),
    .s_axis_dividend_tready (dividend_ready),
    .s_axis_dividend_tvalid (div_valid),
    // M_AXIS_DOUT
    .m_axis_dout_tdata      (div_result),
    .m_axis_dout_tvalid     (div_done)
);
```

图 1：除法器实例化

执行 s_axis_dividend_tdata/s_axis_divisor_tdata，将商放在 m_axis_dout_tdata 的高位，余数放在低位。

2、接口定义

这一部分的核心是 ready，valid 和 done 三种信号的握手，其余信号省略。

当时钟上升沿来临时，将 valid 置为 1，发起握手请求；随后 ready 信号也会拉高（周期性拉高），在下一个时钟上升沿时，两者均为高则完成握手，data 传入 IP 核，进行运算。

最后，当 div_done 拉高时，表明计算结束，阻塞放开，流水线启动。

3、功能描述

调用 IP 核完成除法。

（三）重要模块 2 设计：跳转指令控制信号

1、工作原理

跳转指令的控制信号如图 2 所示：

```
assign rj_eq_rd = (rj_value == rkd_value);
assign rj_l_rd =  (~rj_value[31] && rkd_value[31]) ? 0
                : (rj_value[31] && ~rkd_value[31]) ? 1
                : rj_l_rd_u;
assign rj_ge_rd= ~rj_l_rd;
assign rj_l_rd_u= (rj_value < rkd_value);
assign rj_ge_rd_u= ~rj_l_rd_u;

assign br_taken = (   inst_beq  &&  rj_eq_rd
                   || inst_bne  && !rj_eq_rd
                   || inst_blt  &&  rj_l_rd
                   || inst_bge  &&  rj_ge_rd
                   || inst_bltu &&  rj_l_rd_u
                   || inst_bgeu &&  rj_ge_rd_u
                   || inst_jirl
                   || inst_bl
                   || inst_b
                  ) && ds_valid;
```

图 2：跳转指令控制信号

本实验新增了四条跳转指令，跳转逻辑分别是无符号小于、无符号大于等于、有符号小于、有符号大于等于。显然只需实现两个小于的判断，再对结果取反即可获得两个大于等于的取值。而有符号小于与无符号小于的比较方式也有所不同；小于号<可以直接对两个数进行无符号比较，但对于有符号数则无法这样直接比较。一种方法是采用 signed 标识符，另一种是考虑两个数字的正负分情况讨论。稳妥起见，这里采用后者实现，得到了四个新增信号的赋值逻辑，进一步更改 br_taken 和 br_target 的逻辑即可实现相应跳转。

2、功能描述

通过跳转控制信号判断当前指令是否跳转，完成四条跳转指令的新增。

（四）重要模块 3 设计：访存指令相关计算

1、工作原理

load 类型的指令在从数据 RAM 得到结果后，需要根据指令的类型和访存地址的低二位确定要存回寄存器堆的数据。为此，新增 ld_vaddr 信号指示访存地址的后二位，同时生成 ld.b，ld.bu，ld.h，ld.hu，ld.w 的结果，再通过指令控制信号进行多路选择，将相应的结果存入 mem_result 中。该过程如下图所示：

```
assign ld_vaddr = ms_alu_result[1:0];
assign ld_b_bu_sel = (ld_vaddr == 2'b00) ? data_sram_rdata[ 7: 0] :
                     (ld_vaddr == 2'b01) ? data_sram_rdata[15: 8] :
                     (ld_vaddr == 2'b10) ? data_sram_rdata[23:16] :
                                           data_sram_rdata[31:24] ;
assign ld_b_res  = {{24{ld_b_bu_sel[7]}}, ld_b_bu_sel};     // sign-extension(signed number)
assign ld_bu_res = {{24{1'b0}}, ld_b_bu_sel};               // zero-extension(unsigned number)
assign ld_h_hu_sel = (ld_vaddr == 2'b00) ? data_sram_rdata[15: 0] :
                                           data_sram_rdata[31:16] ;
assign ld_h_res  = {{16{ld_h_hu_sel[15]}}, ld_h_hu_sel};    // sign-extension(signed number)
assign ld_hu_res = {{16{1'b0}}, ld_h_hu_sel};               // zero-extension(unsigned number)
assign mem_result = (ms_ld_op[0]) ? ld_b_res  :
                    (ms_ld_op[1]) ? ld_bu_res :
                    (ms_ld_op[2]) ? ld_h_res  :
                    (ms_ld_op[3]) ? ld_hu_res :
                                    data_sram_rdata;
```

图 3：ld 指令相关计算过程

store 类型的指令在从寄存器堆得到结果后，需要根据指令的类型和访存地址的低二位确定存入数据 RAM 的内容。为此，新增 st_vaddr 信号指示访存地址的后二位，同时生成 mem_write_strb 信号，再根据 mem_write_strb 信号截取相应的字节，将结果存入 mem_write_data 中。该过程如下图所示：

```
assign st_vaddr = es_alu_result[1:0];
assign mem_write_strb = (es_ld_st_op[5] && st_vaddr == 2'b00) ? 4'b0001 :
                        (es_ld_st_op[5] && st_vaddr == 2'b01) ? 4'b0010 :
                        (es_ld_st_op[5] && st_vaddr == 2'b10) ? 4'b0100 :
                        (es_ld_st_op[5] && st_vaddr == 2'b11) ? 4'b1000 :
                        (es_ld_st_op[6] && st_vaddr == 2'b00) ? 4'b0011 :
                        (es_ld_st_op[6] && st_vaddr == 2'b10) ? 4'b1100 :
                                                                4'b1111 ;
assign mem_write_data = es_ld_st_op[5] ? {4{rkd_value[ 7:0]}} :
                        es_ld_st_op[6] ? {2{rkd_value[15:0]}} :
                                         rkd_value[31:0];

assign data_sram_en    = 1'b1;
assign data_sram_wen   = (es_mem_we && es_valid) ? mem_write_strb : 4'h0;
assign data_sram_addr  = es_alu_result;
assign data_sram_wdata = mem_write_data;
```

图 4：st 指令相关计算过程

2、功能描述

通过访存指令具体类型和访存地址低二位计算要存回的数据，然后整体复用之前的数据通路，完成数据写回。

3、注意事项

load 类型的指令在访存阶段计算相应的结果，store 类型的指令则要在执行阶段计算出结果，因为 store 类型的指令需要在访存阶段向 RAM 中写入数据，这意味着计算结果需要提前一个周期拿到。

三、实验过程（50%）

（一）实验流水账

2022.9.30	19：00——23：00	完成 exp10；
2022.10.6	20：30——21：30	实现新增的访存指令；
2022.10.7	21：00——23：00	实现新增的跳转指令；
2022.10.8	14：00——16：00	完成 exp10 部分实验报告的书写；
2022.10.9	20：30——22：00	完成 exp11 跳转指令部分实验报告的书写；
2022.10.11	8：30——9：30	完成 exp11 访存指令部分实验报告的书写。

（二）错误记录

1、错误 1：除法器 ip 核的时序错误

（1）错误现象

div_valid 未能正确拉高，产生错误。波形会一直前进，无法停下。

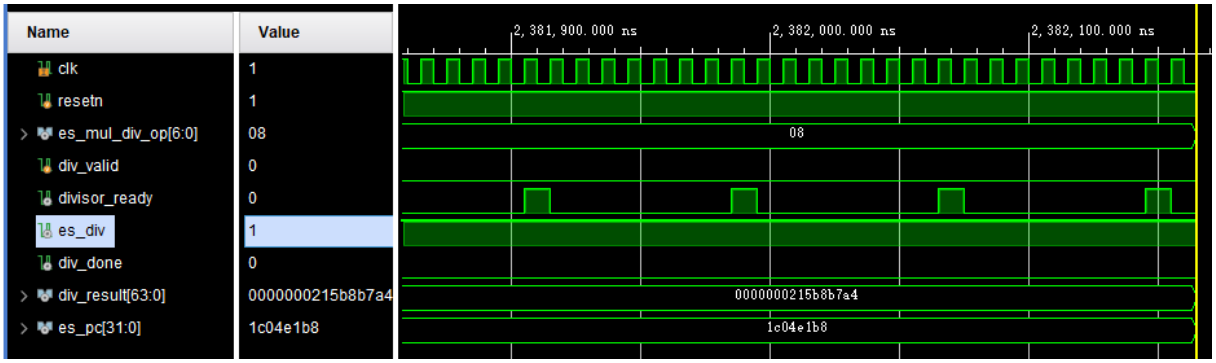

图 5：错误 1 波形截图

（2）分析定位过程

分析 div_valid 的来源，是时序的问题，于是开始用纸笔摆弄时序，发现错误原因。

（3）错误原因

分析 div_valid 的来源，发现是来源于 bus_r（es_mul_div_op），这需要一次 clk 才能将 bus 中的值传入 bus_r，这本是正常的一步。但是由于 div_valid 是由时序控制的，其进入条件为(ds_to_es_valid && es_allowin)，于是会发生下述问题：

```
else if(ds_to_es_valid && es_allowin) begin
    div_valid <= |es_mul_div_op[4:3];//| ds_to_es_bus[152:151];
end
```

图 6：错误 1 出错代码

在时钟上升沿来临时，ID 阶段的 div 指令将进入 EXE 阶段，此时，es_ready_go 的逻辑是组合的，于是在 clk 后，es_mul_div_op 的值成功从 bus_r 中读了出来，但是此时 es_allowin 也同时被拉低，于是 div_valid 的值无法发生变化，不能拉高，最终导致错误。

（4）修正效果

将其直接连上 ds_to_es_bus 即可，这样便可以解决时序的问题。

（5）归纳总结

对于 axis 总线标准以及时序跳变不清晰。

2、错误 2：访存指令的控制信号没有更新

（1）错误现象

新增 load 类型指令写回阶段写回的数据不正确。

（2）分析定位过程

检查 load 类型指令计算结果，发现 mem_result 计算正确，说明数据计算模块设计未出现问题。但 mem_final_result 信号却传回错误。检查波形，发现 mem_final_result 选取了 ms_alu_result 而不是 mem_result 作为要写回的数据。这说明错误发生在控制信号 ms_res_from_mem 中。

沿流水线追溯，找到 ID 阶段相关控制信号的定义上，发现定义中只考虑了 inst_ld_w 信号，而没有考虑新增的四个 load 类型信号。

（3）错误原因

只增加了访存信号结果计算模块，而没有相应地修改 ID 阶段的各个控制信号。

（4）修正效果

增加其它四个信号，并相应地修改其它与 load 和 store 信号有关的控制信号后，结果正确，并通过所有检查点。

四、实验总结

小组合作要搞好分工，另外还要理解好小组使用的代码。