实验 lab6-lab7 报告

学号 2018K8009929030
姓名 热伊莱·图尔贡
箱子号 74

一、实验任务（10%）

1. Lab6：添加指令，包括算术逻辑运算类指令 slti、sltui、andi、ori、xori、sll、srl、sra、pcaddu12i，乘除运算类指令 mul.w、mulh.w、mulh.wu、div.w、mod.w、div.wu、mod.wu。
2. Lab7：添加更多的指令，包括转移指令 blt、bge、bltu、 bgeu，访存指令 ld.b、ld.h、ld.bu、ld.hu、st.b、st.h。

二、实验设计（40%）

1. **slti 和 sltui 指令的添加：** slti 和 slt 指令对于两个源操作数的比较运算以及写回的目的寄存器是完全一致的，且 slti 和 addi.w 指令的操作数来源是一样的。这就意味着 slti 指令的处理一方面可以复用 slt 指令在执行、访存和写回流水阶段的数据通路，另一方面可以复用 addi.w 指令在译码流水阶段的数据通路。相应地，slti 指令在上述各阶段所需的控制信号也与所复用数据通路对应指令的控制信号一致。采用相同的思路， sltui 指令的处理一方面可以复用 sltu 指令在执行、访存和写回流水阶段的数据通路，另一方面可以复用 addi.w 指令在译码流水阶段的数据通路。相应地，slti 指令在上述各阶段所需的控制信号也与所复用数据通路对应指令的控制信号一致。
2. **andi、ori 和 xori 指令的添加：** andi、ori 和 xori 指令可以复用 and、or 和 xor 指令在执行、访存和写回阶段的数据通路。相应地，这些指令在这些流水阶段所需的控制信号与所复用数据通路对应指令的控制信号一致。不过，andi、ori 和 xori 的第二个源操作数是对 12 位立即数 ui12 进行零扩展，没有现成的数据通路可以复用，需要对译码阶段的数据通路进行调整，在生成第二个源操作数的多路选择器中添加一个 “指令码立即数域 ui12 零扩展至 32 位” 的输入。由于生成第二个源操作数的多路选择器的规格发生了变化，因此不仅是 andi、ori 和 xori，原先已实现的指令也要针对这个更新规格的多路选择器生成正确的控制信号。
3. **sll.w、srl.w 和 sra.w 指令的添加：** sll.w、srl.w、sra.w 指令在执行、访存、写回阶段的数据通路可以分别复用 slli.w、srli.w 和 srai.w 指令的数据通路，而这三条指令在译码阶段的数据通路可以复用 add.w 指令的数据通路。余下的工作就是将这些指令在各阶段的控制信号置为与其所复用数据通路的指令一致。
4. **pcaddu12i 指令的添加：** pcaddu12i 指令一方面可以复用 add.w 指令在执行（不包含源操作数准备）、

访存和写回流水阶段的数据通路，另一方面可以分别复用 bl 和 lu12i.w 指令在译码和执行（其中的源操作数准备）流水阶段的数据通路。相应地，pcaddu12i 指令所需的控制信号也与所复用数据通路对应指令的控制信号一致。

5. **乘除法指令的添加：**乘除法指令，两个源操作数均分别来自于 rj 和 rk 寄存器，计算的结果都是写入到 rd 寄 存器，与 add.w 指令比较可知其相同，因此在译码、写回级流水阶段可以复用 add.w 的数据通 路并生成相同的控制信号。数据通路中主要的设计调整是增加乘、除运算部件。我们采用调用 Xilinx IP 的方法实现乘、除法运算部件的设计。
6. **转移指令的添加：**通过分析 blt、bge、bltu 和 bgeu 指令的功能定义，可以得知它们的功能与 beq、bne 非常相似，区别仅在于是否跳转的判断条件。因此添加 blt、bge、bltu 和 bgeu 指令只需要对流水线中已有的转移指令是否跳转的判断逻辑进行扩展即可。
7. **访存指令的添加：**这四条指令在译码、执行、写回级的数据通路、控制逻辑可以完全复用 ld.w 指令的设计实现。

（一）总体设计思路

本次实验是添加三类指令：运算类指令，转移类指令，访存指令。因此我们将其分为三个模块，逐个完善。添加指令的主要过程为译码，加入已有数据通路/创建新的数据通路。

（二）重要模块 1 设计：除法模块

调用 Xilinx IP 的方法实现除法运算部件的设计。

1、工作原理

对于两个输入通道和一个输出通道，每个通道都有一对 tvalid、tready 信号。这是一对 “握手” 控制信号， tvalid 是请求信号，tready 是应答信号。在时钟上升沿到来时，如果采样得到 tvalid 和 tready 都等于 1，则请求发起方和接收方之间完成一次成功的握手。如果我们假想接收方有一组触发器缓存，那么所谓的成功握手是指发送方的数据写入接收方的缓存中，也就是在握手成功的这个上升沿之后，触发器缓存会变为发送方的数据。

2、功能描述

```
always @(posedge clk) begin
    if(div_result_valid) begin
        div_clear <= 1'b0;
    end
    else if(divisor_tready && divisor_tvalid && dividend_tready && dividend_tvalid) begin
        div_clear <= 1'b1;
    end
end

always @(posedge clk) begin
    if(divu_result_valid) begin
        divu_clear <= 1'b0;
    end
    else if(divisor_tready_u && divisor_tvalid_u && dividend_tready_u && dividend_tvalid_u) begin
        divu_clear <= 1'b1;
    end
end
```

图 1　除法器握手信号

（三）重要模块 2 设计：跳转指令

新增指令并译码，并相应的更新数据通路。

1、工作原理

跳转指令的添加基本于计组实验课一致。

2、功能描述

```
//lab7
wire [31:0] adder_res;
wire        carry;
assign {carry, adder_res} = {1'b0, rj_value} + {1'b0, ~rkd_value} + 1'b1;
assign rj_lt_rd  = (rj_value[31] & ~rkd_value[31])
                 | ((rj_value[31] ~^ rkd_value[31]) & adder_res[31]);
assign rj_ltu_rd = ~carry;

assign rj_eq_rd  = (rj_value == rkd_value);

assign br_taken = (   inst_beq  &&  rj_eq_rd
                   || inst_bne  && !rj_eq_rd
                   || inst_blt  &&  rj_lt_rd
                   || inst_bge  && !rj_lt_rd
                   || inst_bltu &&  rj_ltu_rd
                   || inst_bgeu && !rj_ltu_rd
                   || inst_jirl
                   || inst_bl
                   || inst_b
                  ) && ds_valid;
assign br_target = (inst_beq || inst_bne || inst_bl   || inst_b ||
                    inst_blt || inst_bge || inst_bltu || inst_bgeu) ? (ds_pc + br_offs) :
                                                   /*inst_jirl*/ (rj_value + jirl_offs);
```

图 2　跳转指令

根据新增指令更改 rj,rk,rd_read：

```
assign rk_read=inst_add_w|inst_sub_w|inst_slt|inst_sltu|inst_nor|inst_and|inst_or
|inst_xor|inst_mul_w|inst_mulh_w|inst_mulh_wu|inst_div_w|inst_div_wu
|inst_mod_w|inst_mod_wu|inst_sll_w|inst_srl_w|inst_sra_w;
assign rj_read=~inst_lu12i_w & ~inst_b & ~inst_bl & ~inst_pcaddu12i &~ inst_blt &~ inst_bge &~ inst_bltu &~ inst_bgeu;
assign rd_read=inst_beq | inst_bne|inst_st_w|inst_blt|inst_bge|inst_bltu|inst_bgeu;
```

图 3　跳转指令

（四）重要模块 2 设计：访存指令

1、工作原理

访存指令的添加基本于计组实验课一致。

2、功能描述

(1) 实现 st.b 和 st.h 指令的关键在于如何在一块 4 字节宽的 RAM 上完成字 节或半字的写入，这可以通过 RAM 的字节写使能来实现。所谓字节写使能，就是 RAM 每一项 （或行）的每个字节都有自己单独的写使能。可知，在实现 st.b 指令写数据 RAM 的时候，如果地 址最低两位等于 0b00，那么字节写使能就是 0b0001；如果地址最低两位等于 0b01，那么字节写使能就是 0b0010……在实现 st.h 指令的时候，如果地址的最低两位等于 0b00，那么字节写使能就 是 0b0011。对于 st.w，字节写使能恒为 0b1111。

(2) 实现 ld.b、ld.h、ld.bu、ld.hu 指令。由地址的末两位生成 load_sel 信号。Lb.lbu 指令通过地址的后两位来决定取出 32 位中的哪 8 位；Lh,lhu 指令通过 sel[0]来判断取出 32 位中的哪 16 位。

```
wire [3:0]  load_sel;
wire [31:0] lb_data;
wire [31:0] lbu_data;
wire [31:0] lh_data;
wire [31:0] lhu_data;
wire [31:0] load_data;

decoder_2_4  u_dec_ld(.in(ms_alu_result[1:0]), .out(load_sel));

assign lb_data   = {32{  load_sel[0]}} & {{24{mem_result[7]}}, mem_result[7:0]}
                 | {32{  load_sel[1]}} & {{24{mem_result[15]}}, mem_result[15:8]}
                 | {32{  load_sel[2]}} & {{24{mem_result[23]}}, mem_result[23:16]}
                 | {32{  load_sel[3]}} & {{24{mem_result[31]}}, mem_result[31:24]};
assign lbu_data  = {32{  load_sel[0]}} & {24'b0, mem_result[7:0]}
                 | {32{  load_sel[1]}} & {24'b0, mem_result[15:8]}
                 | {32{  load_sel[2]}} & {24'b0, mem_result[23:16]}
                 | {32{  load_sel[3]}} & {24'b0, mem_result[31:24]};
assign lh_data   = {32{  load_sel[0]}} & {{16{mem_result[15]}}, mem_result[15:0]}
                 | {32{ ~load_sel[0]}} & {{16{mem_result[31]}}, mem_result[31:16]};
assign lhu_data  = {32{  load_sel[0]}} & {16'b0, mem_result[15:0]}
                 | {32{ ~load_sel[0]}} & {16'b0, mem_result[31:16]};
assign load_data = {32{ms_load_op[0]}} & mem_result
                 | {32{ms_load_op[1]}} & lb_data
                 | {32{ms_load_op[2]}} & lh_data
                 | {32{ms_load_op[3]}} & lbu_data
                 | {32{ms_load_op[4]}} & lhu_data;
assign ms_final_result = (ms_load_op != 5'b0) ? load_data
                                              : ms_alu_result;
```

图 4　ld 指令

```
//lab7
//st
wire [31:0] st_data;
wire [3:0]  st_strb;
wire [3:0]  st_sel;
wire op_st_b = es_st_op[1];
wire op_st_h = es_st_op[2];
decoder_2_4 decoder_st(
    .in  (es_alu_result[1:0]),
    .out (st_sel)
);

assign st_strb = op_st_h ? (st_sel[0]? 4'b0011 : 4'b1100) :
                 op_st_b ?  st_sel                        :
                 4'b1111;
assign st_data = op_st_b ? {4{es_rkd_value[ 7:0]}} :
                 op_st_h ? {2{es_rkd_value[15:0]}} :
                 es_rkd_value[31:0];

assign es_mem_we = (es_st_op != 3'b0);
assign data_sram_en    = (es_res_from_mem || es_mem_we) && es_valid;
assign data_sram_wen   = es_mem_we ? 4'hf : 4'h0;
assign data_sram_addr  = {es_alu_result[31:2],2'b0};
assign data_sram_wdata = st_data; //es_rkd_value;
```

图 5　st 指令

三、实验过程（50%）

（一）实验流水账

2021.11.9 之前完成 lab6。

2021.11.30 完成了 lab7。

（二）错误记录

Debug 过程忘记截图了。

四、实验总结（可选）