中国科学院大学
《计算机体系结构（研讨课）》实验报告

小组成员姓名 孙广润蔡合森刘钰

实验项目编号 实践任务 10, 11 实验名称 添加用户态指令设计专题

一、 算数逻辑类与乘除法运算指令的添加

对于算数逻辑类的指令，对于可以复用数据通路的指令，如 sltui, 则复用 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；不能复用数据通路的 andi、ori 和 xori ，就对第二个源操作数是对 12 位立即数 ui12 进行零扩展

对于乘除法运算类指令的添加, 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.

- **myCPU 修改部分**。

```
assign inst_slti  = op_31_26_d[6'h00] & op_25_22_d[4'h8];
assign inst_sltiu = op_31_26_d[6'h00] & op_25_22_d[4'h9];
assign inst_andi  = op_31_26_d[6'h00] & op_25_22_d[4'hd];
assign inst_ori   = op_31_26_d[6'h00] & op_25_22_d[4'he];
assign inst_xori  = op_31_26_d[6'h00] & op_25_22_d[4'hf];
assign inst_sll_w = op_31_26_d[6'h00] & op_25_22_d[4'h0] & op_21_20_d[2'h1] & op_19_15_d[5'h0e];
assign inst_srl_w = op_31_26_d[6'h00] & op_25_22_d[4'h0] & op_21_20_d[2'h1] & op_19_15_d[5'h0f];
assign inst_sra_w = op_31_26_d[6'h00] & op_25_22_d[4'h0] & op_21_20_d[2'h1] & op_19_15_d[5'h10];
assign inst_pcaddu12i= op_31_26_d[6'h07] & ~inst[25];
\\乘除法运算类指令有效信号
assign inst_mul_w = op_31_26_d[6'h00] & op_25_22_d[4'h0] & op_21_20_d[2'h1] & op_19_15_d[5'h18];
assign inst_mulh_w = op_31_26_d[6'h00] & op_25_22_d[4'h0] & op_21_20_d[2'h1] & op_19_15_d[5'h19];
assign inst_mulh_wu = op_31_26_d[6'h00] & op_25_22_d[4'h0] & op_21_20_d[2'h1] & op_19_15_d[5'h1a];
assign inst_div_w = op_31_26_d[6'h00] & op_25_22_d[4'h0] & op_21_20_d[2'h2] & op_19_15_d[5'h00];
assign inst_mod_w = op_31_26_d[6'h00] & op_25_22_d[4'h0] & op_21_20_d[2'h2] & op_19_15_d[5'h01];
assign inst_div_wu = op_31_26_d[6'h00] & op_25_22_d[4'h0] & op_21_20_d[2'h2] & op_19_15_d[5'h02];
assign inst_mod_wu = op_31_26_d[6'h00] & op_25_22_d[4'h0] & op_21_20_d[2'h2] & op_19_15_d[5'h03];
//add new signals and needed results
wire      need_div;
wire      div_unsigned;
wire      div_signed;
wire      get_div_or_mod;
wire [63:0] sdiv_result;
wire [63:0] udiv_result;
wire [31:0] EX_final_result;
//add regs used in the following percedure
reg [14:0] alu_op_r;
reg      need_div_r;
reg      div_unsigned_r;
reg      div_signed_r;
reg      get_div_or_mod_r;
```

```
assign ID_readygo = !need_div_r;//阻塞除法
```

- **ALU 修改部分**。

```
wire op_mul; // multiplication and get low
wire op_mulh; // signed multiplication and get high
wire op_mulhu; // unsigned multiplication and get high

assign op_mul = alu_op[12];
assign op_mulh = alu_op[13];
assign op_mulhu= alu_op[14];

// MUL result
wire [32:0] mul_a, mul_b;
wire [65:0] mul_result;
assign mul_a = {alu_src1[31]&&!op_mulhu,alu_src1};
assign mul_b = {alu_src2[31]&&!op_mulhu,alu_src2};
assign mul_result = mul_a * mul_b;

assign alu_result = ({32{op_add|op_sub}} & add_sub_result)
//add new results
                | ({32{op_srl|op_sra}} & sr_result)
                | ({32{op_mul    }} & mul_result[31:0])
                | ({32{op_mulh|op_mulhu}} & mul_result[63:32]);
//修改前递的值，修改为EX阶段的值
assign rj_pro = (rj == dest_EX_ID) ? EX_final_result :
              (rj == dest_MEM_ID) ? data_sram_addr_MEM :
              final_result ;

assign rk_pro = (rk == dest_EX_ID) ? EX_final_result :
              (rk == dest_MEM_ID) ? data_sram_addr_MEM :
              final_result ;

assign rd_pro = (rd == dest_EX_ID) ? EX_final_result :
              (rd == dest_MEM_ID) ? data_sram_addr_MEM :
              final_result ;
//添加新的信号
assign need_div = inst_div_w | inst_mod_w | inst_div_wu | inst_mod_wu;
assign div_signed = inst_div_w | inst_mod_w;
assign div_unsigned = inst_div_wu | inst_mod_wu;
assign get_div_or_mod = inst_div_w | inst_div_wu;

assign data_sram_en_ID = inst_ld_w || inst_st_w;
assign data_sram_we_ID = {4{mem_we && valid}};
assign data_sram_wdata_ID = rkd_value;

//将alu及除法器输入保存到EX阶段并使用
always @(posedge clk) begin
   if(reset) begin
```

```
        alu_src1_r <= 32'h0;
        alu_src2_r <= 32'h0;
        alu_op_r  <= 15'h0;
        div_signed_r <= 1'b0;
        div_unsigned_r <= 1'b0;
        get_div_or_mod_r <= 1'b0;
    end
    else if(IF_valid && ID_allowin && IF_readygo)begin
        alu_src1_r <= alu_src1;
        alu_src2_r <= alu_src2;
        alu_op_r  <= alu_op;
        div_signed_r <= div_signed;
        div_unsigned_r <= div_unsigned;
        get_div_or_mod_r <= get_div_or_mod;
    end
end
```

- **除法器模块**。

这里采用了提供的现成 IP，不需要自己设计。

```
alu u_alu(// alu进行运算
    .alu_op    (alu_op_r  ),
    .alu_src1  (alu_src1_r ),
    .alu_src2  (alu_src2_r ),
    .alu_result (alu_result)
    );

div_gen_signed u_div_gen_signed(// 进行符号除法运算
    .aclk                (clk       ),
    .s_axis_divisor_tdata (alu_src1_r ),
    .s_axis_divisor_tvalid (sdiv_sor_valid),
    .s_axis_divisor_tready (sdiv_sor_ready),
    .s_axis_dividend_tdata (alu_src2_r ),
    .s_axis_dividend_tvalid (sdiv_dend_valid),
    .s_axis_dividend_tready (sdiv_dend_ready),
    .m_axis_dout_tdata   (sdiv_result ),
    .m_axis_dout_tvalid  (sdiv_out_valid)
    );

div_gen_unsigned u_div_gen_unsigned(// 进行无符号除法运算
    .aclk                (clk       ),
    .s_axis_divisor_tdata (alu_src1_r ),
    .s_axis_divisor_tvalid (udiv_sor_valid),
    .s_axis_divisor_tready (udiv_sor_ready),
    .s_axis_dividend_tdata (alu_src2_r ),
    .s_axis_dividend_tvalid (udiv_dend_valid),
    .s_axis_dividend_tready (udiv_dend_ready),
    .m_axis_dout_tdata   (udiv_result ),
    .m_axis_dout_tvalid  (udiv_out_valid)
    );
```

二、 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。

```
//分支指令
wire      inst_blt;
wire      inst_bge;
wire      inst_bltu;
wire      inst_bgeu;
//新添加分支指令有效信号
assign inst_blt  = op_31_26_d[6'h18];
assign inst_bge  = op_31_26_d[6'h19];
assign inst_bltu = op_31_26_d[6'h1a];
assign inst_bgeu = op_31_26_d[6'h1b];
//添加新的情况
assign need_si16 = inst_jirl | inst_beq | inst_bne | inst_blt | inst_bge | inst_bltu | inst_bgeu;

/******** 分支判断块 ********/
wire rj_lt_rds;
wire rj_lt_rdu;
assign rj_lt_rds = (rj_value[31]==1'b1&&rkd_value[31]==1'b0) ? 1'b1 :
(rj_value[31]==1'b0&&rkd_value[31]==1'b1) ? 1'b0 :
(rj_value[31]==1'b0&&rkd_value[31]==1'b0)? (rj_value < rkd_value) :
(rj_value[31]==1'b1&&rkd_value[31]==1'b1) ? (rj_value[30:0] < rkd_value[30:0]) : 1'b0;

assign rj_lt_rdu = (rj_value < rkd_value);
//更新br_taken与br_target的情况
assign br_taken = ( inst_beq && rj_eq_rd
                || inst_bne && !rj_eq_rd
                || inst_blt && rj_lt_rds
                || inst_bge && !rj_lt_rds
                || inst_bltu && rj_lt_rdu
                || inst_bgeu && !rj_lt_rdu
                || inst_jirl
                || inst_bl
                || inst_b
               ) && IF_valid;
assign br_target = (inst_beq || inst_bne || inst_bl || inst_b || inst_blt || inst_bge || inst_bltu
    || inst_bgeu) ? (pc_ID + br_offs) :
                                          /*inst_jirl*/ (rj_value + jirl_offs);
```

三、 访存指令的添加

- **ld.b、ld.h、ld.bu、ld.hu 指令的添加**。

ld.b、ld.h、ld.bu、ld.hu 指令的功能定义并将其与 ld.w 的定义进行比较，可知：它们计算虚地址的操作数来源、地址计算方法、虚实地址映射的规则是完全一样的。它们得到的访存结果都是写回第 rd 项寄存器中。它们和 ld.w 指令的差异仅在于从内存取回的数据位宽不同。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。只需要从数据 RAM 输出结果中选择所需内容，将选取内容扩展至 32 位即可• **st.b、st.h 指令的添加**。

通过 RAM 的字节写使能来实现，st.b 和 st.h 指令的 4 字节宽的 RAM 上字节或半字的写入，

```
//访存指令
wire      inst_ld_b;
wire      inst_ld_h;
wire      inst_ld_bu;
wire      inst_ld_hu;
wire      inst_st_b;
wire      inst_st_h;
//数据通路
wire [ 2:0] data_sram_addroffset;
wire [31:0] data_sram_wdata_ID;
wire [ 2:0] data_sram_ld_tag;
wire [31:0] data_sram_rdata_off;
wire [31:0] st_data;

reg [ 2:0] data_sram_ld_tag_EX;
reg [ 2:0] data_sram_ld_tag_MEM;

//新添加访存指令有效信号
assign inst_ld_b = op_31_26_d[6'h0a] & op_25_22_d[4'h0];
assign inst_ld_h = op_31_26_d[6'h0a] & op_25_22_d[4'h1];
assign inst_st_b = op_31_26_d[6'h0a] & op_25_22_d[4'h4];
assign inst_st_h = op_31_26_d[6'h0a] & op_25_22_d[4'h5];
assign inst_ld_bu = op_31_26_d[6'h0a] & op_25_22_d[4'h8];
assign inst_ld_hu = op_31_26_d[6'h0a] & op_25_22_d[4'h9];

assign alu_op[ 0] = inst_add_w | inst_addi_w | inst_ld_b | inst_ld_h | inst_ld_bu | inst_ld_hu |
    inst_ld_w
                | inst_st_b | inst_st_h | inst_st_w | inst_jirl | inst_bl | inst_pcaddu12i;

assign src2_is_imm = inst_slli_w |
                     inst_srli_w |
                     inst_srai_w |
                     inst_addi_w |
                     inst_ld_b |
                     inst_ld_h |
                     inst_ld_bu |
                     inst_ld_hu |
                     inst_ld_w |
                     inst_st_b |
```

```
                    inst_st_h |
                    inst_st_w |
                    inst_lu12i_w|
                    inst_jirl |
                    inst_bl   |
                    inst_slti |
                    inst_sltiu |
                    inst_andi |
                    inst_ori  |
                    inst_xori |
                    inst_pcaddu12i;


assign data_sram_en_ID = inst_ld_b | inst_ld_h | inst_ld_bu | inst_ld_hu | inst_ld_w | inst_st_b |
    inst_st_h | inst_st_w;
assign data_sram_we_ID = {4{inst_st_w}} | {2'b00, {2{inst_st_h}}} | {3'b000, inst_st_b};
assign data_sram_wdata_ID = inst_st_b ? {4{rkd_value[ 7:0]}} :
                        inst_st_h ? {2{rkd_value[15:0]}} :
                        rkd_value;
assign data_sram_ld_tag = {{inst_ld_b | inst_ld_bu}, {inst_ld_h | inst_ld_hu}, {inst_ld_bu |
    inst_ld_hu}};// {byte_en, half_en, unsigned_en}
//设置访存信号
assign data_sram_addr = {data_sram_addr_MEM[31:2], 2'b00};//对齐地址
assign data_sram_addroffset = data_sram_addr_MEM[1:0];//访存偏移
//选择RAM的字节
assign data_sram_rdata_off = data_sram_rdata >> (data_sram_addroffset * 8);
assign mem_result = data_sram_ld_tag_MEM[2]? {{24{data_sram_rdata_off[7] &
    ~data_sram_ld_tag_MEM[0]}}, data_sram_rdata_off[7:0]} :
                data_sram_ld_tag_MEM[1]? {{16{data_sram_rdata_off[15] &
                    ~data_sram_ld_tag_MEM[0]}}, data_sram_rdata_off[15:0]} :
                data_sram_rdata_off;
```

四、 实验分工

- **实践十**。
  * 刘钰：算术逻辑运算类指令 ‘slti‘, ‘sltui‘, ‘andi‘, ‘ori‘, ‘xori‘, ‘sll‘, ‘srl‘, ‘sra‘, ‘pcaddu12i‘;
  * 孙广润：乘除运算类指令 ‘mul.w‘, ‘mulh.w‘, ‘mulh.wu‘, ‘div.w‘, ‘mod.w‘, ‘div.wu‘, ‘mod.wu‘;
  * 孙广润，蔡合森：debug;
- **实践十一**。

l* 刘钰：转移指令 ‘blt‘, ‘bge‘, ‘bltu‘, ‘bgeu‘;debug;

  * 孙广润：访存指令 ‘ld.b‘, ‘ld.h‘, ‘ld.bu‘, ‘ld.hu‘, ‘st.b‘, ‘st.h‘ debug
  * 蔡合森：报告编写