Lab 4 report

PB22051022王嘉宁

实验目的与内容

> 完善上一次实验设计的 CPU，为其增加分支指令和访存指令的相关功能。

逻辑设计

> 任务 2：搭建 CPU 关键代码段

CPU连接

```
module CPU (···
);
PC my_pc (
    .clk    (clk        ),
    .rst    (rst        ),
    .en     (global_en  ),    // 当 global_en 为高电平时，PC 才会更新，CPU 才会执行指
令。
    .npc    (cur_npc    ),
    .pc     (cur_pc     )
);
assign imem_raddr = cur_pc;
ADD4 adder(
    .a      (cur_pc     ),
    .b      (32'd4      ),
    .c      (pc_add4    )
);
NPC_MUX npc_mux(
    .pc_add4(pc_add4),
    .pc_offset(alu_res),
    .npc_sel(npc_sel),
    .npc(cur_npc)
);
assign cur_inst = imem_rdata;
DECODE decode(
    .inst(cur_inst),
    .alu_op(alu_op),
    .dmem_access(dmem_access),
    .imm(imm),
    .rf_ra0(rf_ra0),
    .rf_ra1(rf_ra1),
    .rf_wa(rf_wa),
    .rf_we(rf_we),
    .rf_wd_sel(rf_wd_sel),
    .alu_src0_sel(alu_src0_sel),
```

```
    .alu_src1_sel(alu_src1_sel),
    .br_type(br_type),
    .dmem_we(dmem_we)
    );
REGFILE regfile(
    .clk(clk),
    .rf_ra0(rf_ra0),
    .rf_ra1(rf_ra1),
    .dbg_reg_ra(debug_reg_ra),
    .rf_wa(rf_wa),
    .rf_we(rf_we),
    .rf_wd(rf_wd),
    .rf_rd0(rf_rd0),
    .rf_rd1(rf_rd1),
    .dbg_reg_rd(debug_reg_rd)
);
MUX mux0(
    .src0(rf_rd0),
    .src1(cur_pc),
    .sel(alu_src0_sel),
    .res(alu_src0)
);
MUX mux1(
    .src0(rf_rd1),
    .src1(imm),
    .sel(alu_src1_sel),
    .res(alu_src1)
);
ALU alu(
    .alu_src0(alu_src0),
    .alu_src1(alu_src1),
    .alu_op(alu_op),
    .alu_res(alu_res)
);
assign dmem_addr = alu_res;
BRANCH branch(
    .br_type(br_type),
    .br_src0(rf_rd0),
    .br_src1(rf_rd1),
    .npc_sel(npc_sel)
);
SLU slu(
    .addr(alu_res),
    .dmem_access(dmem_access),
    .rd_in(dmem_rdata),
    .wd_in(rf_rd1),
    .rd_out(dmem_rd_out),
    .wd_out(dmem_wdata)
);
MUX2 mux2(
    .src0(pc_add4),
```

```
    .src1(alu_res),
    .src2(dmem_rd_out),
    .src3(32'd0),
    .sel(rf_wd_sel),
    .res(rf_wd)
);
endmodule
```

SLU

```
module SLU (···
);
always @(*) begin
    case(dmem_access)
        4'b0001: begin
            rd_out = rd_in;
            wd_out = 0;
            end
        4'b0010: begin
            rd_out = addr[1] ? {{16{rd_in[31]}},rd_in[31:16]} :
{{16{rd_in[15]}},rd_in[15:0]};
            wd_out = 0;
            end
        4'b0011: begin
            rd_out = addr[1] ? (addr[0] ? {{24{rd_in[31]}},rd_in[31:24]} :
{{24{rd_in[23]}},rd_in[23:16]}) : (addr[0] ? {{24{rd_in[15]}},rd_in[15:8]} :
{{24{rd_in[7]}},rd_in[7:0]});
            wd_out = 0;
            end
        4'b0100: begin
            rd_out = addr[1] ? {16'b0,rd_in[31:16]} : {16'b0,rd_in[15:0]};
            wd_out = 0;
            end
        4'b0101: begin
            rd_out = addr[1] ? (addr[0] ? {24'b0,rd_in[31:24]} :
{24'b0,rd_in[23:16]}) : (addr[0] ? {24'b0,rd_in[15:8]} : {24'b0,rd_in[7:0]});
            wd_out = 0;
            end
        4'b0110: begin
            rd_out = 0;
            wd_out = wd_in;
            end
        4'b0111: begin
            rd_out = 0;
            wd_out = addr[1] ? {wd_in[15:0],rd_in[15:0]} :
{rd_in[31:16],wd_in[15:0]};
            end
        4'b1000: begin
            rd_out = 0;
```

```
            wd_out = addr[1] ? (addr[0] ? {wd_in[7:0],rd_in[23:0]} :
{rd_in[31:24],wd_in[7:0],rd_in[15:0]}) : (addr[0] ?
{rd_in[31:16],wd_in[7:0],rd_in[7:0]} : {rd_in[31:8],wd_in[7:0]});
            end
        default: begin
            rd_out = 0;
            wd_out = 0;
            end
    endcase
end

endmodule
```

BRANCH

```
module BRANCH(···
);
always @(*) begin
    case(br_type)
        4'b0011:                                    npc_sel = 2'b01;//jirl
        4'b0100:                                    npc_sel = 2'b01;//b
        4'b0101:                                    npc_sel = 2'b01;//bl
        4'b0110:begin
                if(br_src0 == br_src1)              npc_sel = 2'b01;
                else                                npc_sel = 2'b00;
                end//beq
        4'b0111:begin
                if(br_src0 != br_src1)              npc_sel = 2'b01;
                else                                npc_sel = 2'b00;
                end//bne
        4'b1000:begin
                if($signed(br_src0) < $signed(br_src1))   npc_sel = 2'b01;
                else                                npc_sel = 2'b00;
                end//blt
        4'b1001:begin
                if($signed(br_src0) >= $signed(br_src1))  npc_sel = 2'b01;
                else                                npc_sel = 2'b00;
                end//bge
        4'b1010:begin
                if(br_src0 < br_src1)               npc_sel = 2'b01;
                else                                npc_sel = 2'b00;
                end//bltu
        4'b1011:begin
                if(br_src0 >= br_src1)              npc_sel = 2'b01;
                else                                npc_sel = 2'b00;
                end//bgeu
        default:                                    npc_sel = 2'b00;
    endcase
end
```

```
endmodule
```

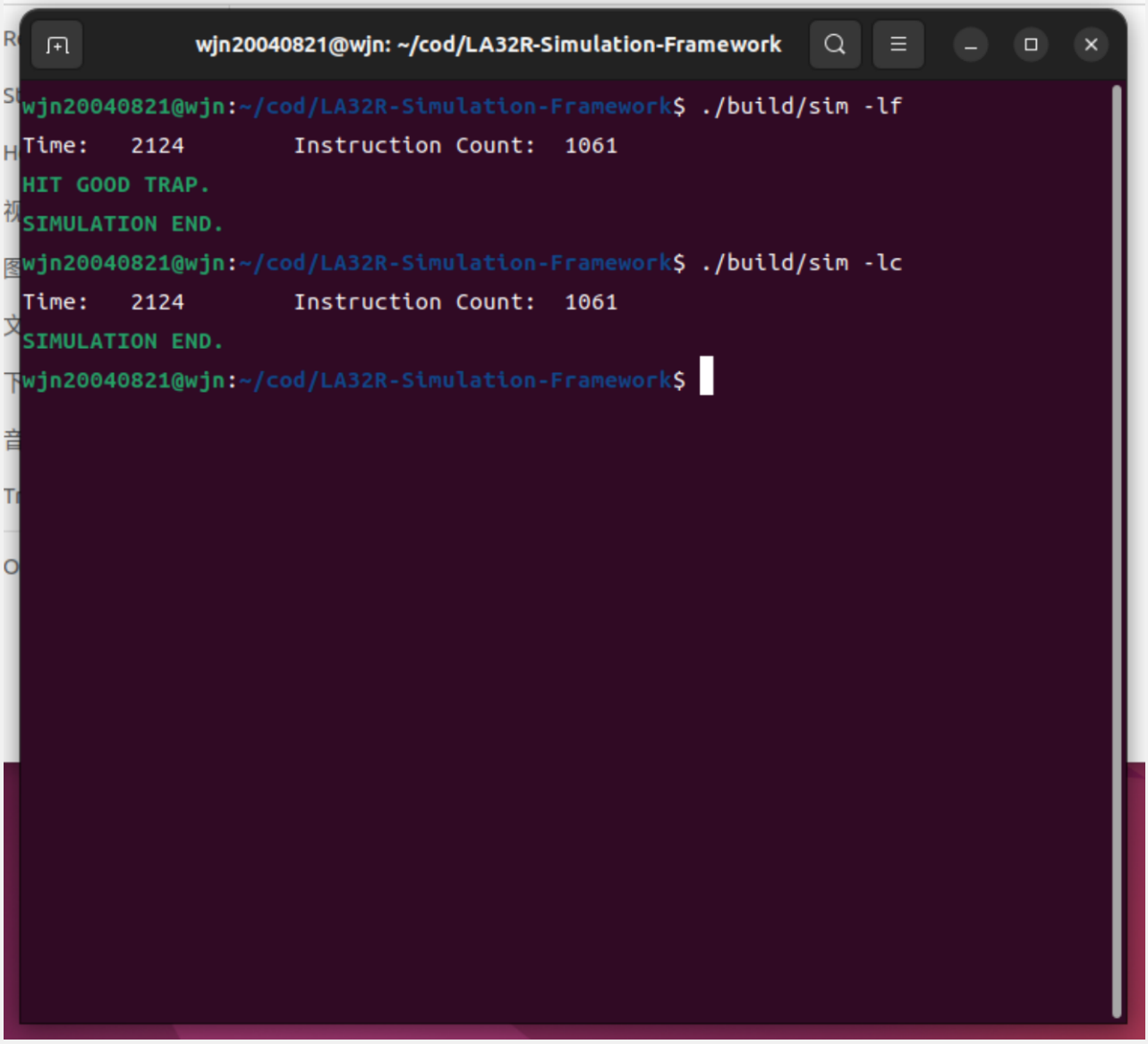
DECODE

```
module DECODE(···
);
always @(*) begin
    else if(inst[31:26] == 6'b001010)begin
        case(inst[25:22])
            4'b0010:begin rf_wa = inst[4:0]; rf_ra0 = inst[9:5]; rf_ra1 =     0;
rf_we = 1; alu_op = 5'b00000; alu_src0_sel = 0; alu_src1_sel = 1; imm =
{{20{inst[21]}},inst[21:10]}; dmem_access = 4'b0001; rf_wd_sel = 2'b10; br_type =
0; dmem_we = 0; end
            4'b0001:begin rf_wa = inst[4:0]; rf_ra0 = inst[9:5]; rf_ra1 =     0;
rf_we = 1; alu_op = 5'b00000; alu_src0_sel = 0; alu_src1_sel = 1; imm =
{{20{inst[21]}},inst[21:10]}; dmem_access = 4'b0010; rf_wd_sel = 2'b10; br_type =
0; dmem_we = 0; end
            4'b0000:begin rf_wa = inst[4:0]; rf_ra0 = inst[9:5]; rf_ra1 =     0;
rf_we = 1; alu_op = 5'b00000; alu_src0_sel = 0; alu_src1_sel = 1; imm =
{{20{inst[21]}},inst[21:10]}; dmem_access = 4'b0011; rf_wd_sel = 2'b10; br_type =
0; dmem_we = 0; end
            4'b1001:begin rf_wa = inst[4:0]; rf_ra0 = inst[9:5]; rf_ra1 =     0;
rf_we = 1; alu_op = 5'b00000; alu_src0_sel = 0; alu_src1_sel = 1; imm =
{{20{inst[21]}},inst[21:10]}; dmem_access = 4'b0100; rf_wd_sel = 2'b10; br_type =
0; dmem_we = 0; end
            4'b1000:begin rf_wa = inst[4:0]; rf_ra0 = inst[9:5]; rf_ra1 =     0;
rf_we = 1; alu_op = 5'b00000; alu_src0_sel = 0; alu_src1_sel = 1; imm =
{{20{inst[21]}},inst[21:10]}; dmem_access = 4'b0101; rf_wd_sel = 2'b10; br_type =
0; dmem_we = 0; end
            4'b0110:begin rf_wa =     0; rf_ra0 = inst[9:5]; rf_ra1 = inst[4:0];
rf_we = 0; alu_op = 5'b00000; alu_src0_sel = 0; alu_src1_sel = 1; imm =
{{20{inst[21]}},inst[21:10]}; dmem_access = 4'b0110; rf_wd_sel = 2'b10; br_type =
0; dmem_we = 1; end
            4'b0101:begin rf_wa =     0; rf_ra0 = inst[9:5]; rf_ra1 = inst[4:0];
rf_we = 0; alu_op = 5'b00000; alu_src0_sel = 0; alu_src1_sel = 1; imm =
{{20{inst[21]}},inst[21:10]}; dmem_access = 4'b0111; rf_wd_sel = 2'b10; br_type =
0; dmem_we = 1; end
            4'b0100:begin rf_wa =     0; rf_ra0 = inst[9:5]; rf_ra1 = inst[4:0];
rf_we = 0; alu_op = 5'b00000; alu_src0_sel = 0; alu_src1_sel = 1; imm =
{{20{inst[21]}},inst[21:10]}; dmem_access = 4'b1000; rf_wd_sel = 2'b10; br_type =
0; dmem_we = 1; end
            default:begin rf_wa = 0; rf_ra0 = 0; rf_ra1 = 0; rf_we = 0; alu_op =
0; alu_src0_sel = 0; alu_src1_sel = 0; imm = 0; dmem_access = 0; rf_wd_sel = 0;
br_type = 0; dmem_we = 0; end
        endcase
    end
    else if(inst[31:30] == 2'b01)begin
        case(inst[29:26])
```

```
            4'b0011:begin rf_wa = inst[4:0]; rf_ra0 = inst[9:5]; rf_ra1 =     0;
rf_we = 1; alu_op = 5'b00000; alu_src0_sel = 0; alu_src1_sel = 1; imm =
{{14{inst[25]}},inst[25:10],2'b0}; dmem_access = 0; rf_wd_sel = 2'b00; br_type =
4'b0011; dmem_we = 0; end
            4'b0100:begin rf_wa =         0; rf_ra0 =         0; rf_ra1 =     0;
rf_we = 0; alu_op = 5'b00000; alu_src0_sel = 1; alu_src1_sel = 1; imm =
{{4{inst[9]}},inst[9:0],inst[25:10],2'b0}; dmem_access = 0; rf_wd_sel = 2'b11;
br_type = 4'b0100; dmem_we = 0; end
            4'b0101:begin rf_wa =      5'b1; rf_ra0 =         0; rf_ra1 =     0;
rf_we = 1; alu_op = 5'b00000; alu_src0_sel = 1; alu_src1_sel = 1; imm =
{{4{inst[9]}},inst[9:0],inst[25:10],2'b0}; dmem_access = 0; rf_wd_sel = 2'b00;
br_type = 4'b0101; dmem_we = 0; end
            4'b0110:begin rf_wa =     0; rf_ra0 = inst[9:5]; rf_ra1 = inst[4:0];
rf_we = 0; alu_op = 5'b00000; alu_src0_sel = 1; alu_src1_sel = 1; imm =
{{14{inst[25]}},inst[25:10],2'b0}; dmem_access = 0; rf_wd_sel = 2'b11; br_type =
4'b0110; dmem_we = 0; end
            4'b0111:begin rf_wa =     0; rf_ra0 = inst[9:5]; rf_ra1 = inst[4:0];
rf_we = 0; alu_op = 5'b00000; alu_src0_sel = 1; alu_src1_sel = 1; imm =
{{14{inst[25]}},inst[25:10],2'b0}; dmem_access = 0; rf_wd_sel = 2'b11; br_type =
4'b0111; dmem_we = 0; end
            4'b1000:begin rf_wa =     0; rf_ra0 = inst[9:5]; rf_ra1 = inst[4:0];
rf_we = 0; alu_op = 5'b00000; alu_src0_sel = 1; alu_src1_sel = 1; imm =
{{14{inst[25]}},inst[25:10],2'b0}; dmem_access = 0; rf_wd_sel = 2'b11; br_type =
4'b1000; dmem_we = 0; end
            4'b1001:begin rf_wa =     0; rf_ra0 = inst[9:5]; rf_ra1 = inst[4:0];
rf_we = 0; alu_op = 5'b00000; alu_src0_sel = 1; alu_src1_sel = 1; imm =
{{14{inst[25]}},inst[25:10],2'b0}; dmem_access = 0; rf_wd_sel = 2'b11; br_type =
4'b1001; dmem_we = 0; end
            4'b1010:begin rf_wa =     0; rf_ra0 = inst[9:5]; rf_ra1 = inst[4:0];
rf_we = 0; alu_op = 5'b00000; alu_src0_sel = 1; alu_src1_sel = 1; imm =
{{14{inst[25]}},inst[25:10],2'b0}; dmem_access = 0; rf_wd_sel = 2'b11; br_type =
4'b1010; dmem_we = 0; end
            4'b1011:begin rf_wa =     0; rf_ra0 = inst[9:5]; rf_ra1 = inst[4:0];
rf_we = 0; alu_op = 5'b00000; alu_src0_sel = 1; alu_src1_sel = 1; imm =
{{14{inst[25]}},inst[25:10],2'b0}; dmem_access = 0; rf_wd_sel = 2'b11; br_type =
4'b1011; dmem_we = 0; end
            default:begin rf_wa = 0; rf_ra0 = 0; rf_ra1 = 0; rf_we = 0; alu_op =
0; alu_src0_sel = 0; alu_src1_sel = 0; imm = 0; dmem_access = 0; rf_wd_sel = 0;
br_type = 0; dmem_we = 0; end
        endcase
    end
end
endmodule
```

测试结果与分析

任务 2：搭建 CPU



上板结果

任务 2：搭建 CPU


uart show>
FPGAOL UART xterm.js 1.1
USTC COD Project
User: R;
---------- CPU ----------
User: RR 0 20;
00000000 1C0005A4 1C0005A4 00000000
00000000 00000000 1C8001B7 7FB437FF
036A0000 CA000000 1C80084A 1C000550
0030365D 00000000 000009CA 00000000
1C00056C 00000000 1C80036A 000009CA
User:
uart pins: cts rts rxd txd
xdc sym: D3 E5 D4 C4
baud rate: 115200
RR 0 20;
input

任务 3：斐波那契数列

任务四

1、假设我们的存储器支持掩码访问，其对应接口如下：

```
module MEM (
    input       [ 0 : 0]              clk,
    input       [ 9 : 0]              a,
    output      [31 : 0]              spo,
    input       [ 0 : 0]              we,
    input       [31 : 0]              d,
    input       [ 3 : 0]              mask
);
```

> 其中，mask 为高电平有效的控制信号，mask[0] 控制当前正在访问的字的最低字节是否有效；mask[3] 控制当前正在访问的字的最高字节是否有效。例如，如果 a = 0x4，we = 1，d = 0x12345678，mask=4'B0110，则数据存储器对应的操作为

```
M[0x4] <- M[0x4]
M[0x5] <- 0x56
M[0x6] <- 0x34
M[0x7] <- M[0x7]
```

> 读操作则会将 mask 为 0 的字节置为 0。例如，如果 a = 0x4，we = 0，mask=4'B0110，则读出的结果为

```
{8'B0, M[0x6], M[0x5], 8'B0}
```

> 请重新设计 SL_UNIT 单元。你可以自行添加相关模块所需要的端口。注意：本题的前提是我们假设存在这样的存储器 IP 核，你并不需要实现这个 IP 核，也不需要仿真或上板，仅阐述 SL_UNIT 单元的设计方案即可。

引入mask端口，在检测dmem_access分支的内部检测mask[0] mask[1] mask[2] mask[3]，若某一位为1，则在拼接的时候将对应的8位置零即可

> 2、请指出本次实验的 CPU 中可能的关键路径。

PC-DECODER-REGFILE-MUX-ALU-SLU-MUX-REGFILE

总结

> 完善了上一次实验设计的 CPU，为其增加分支指令和访存指令的相关功能。