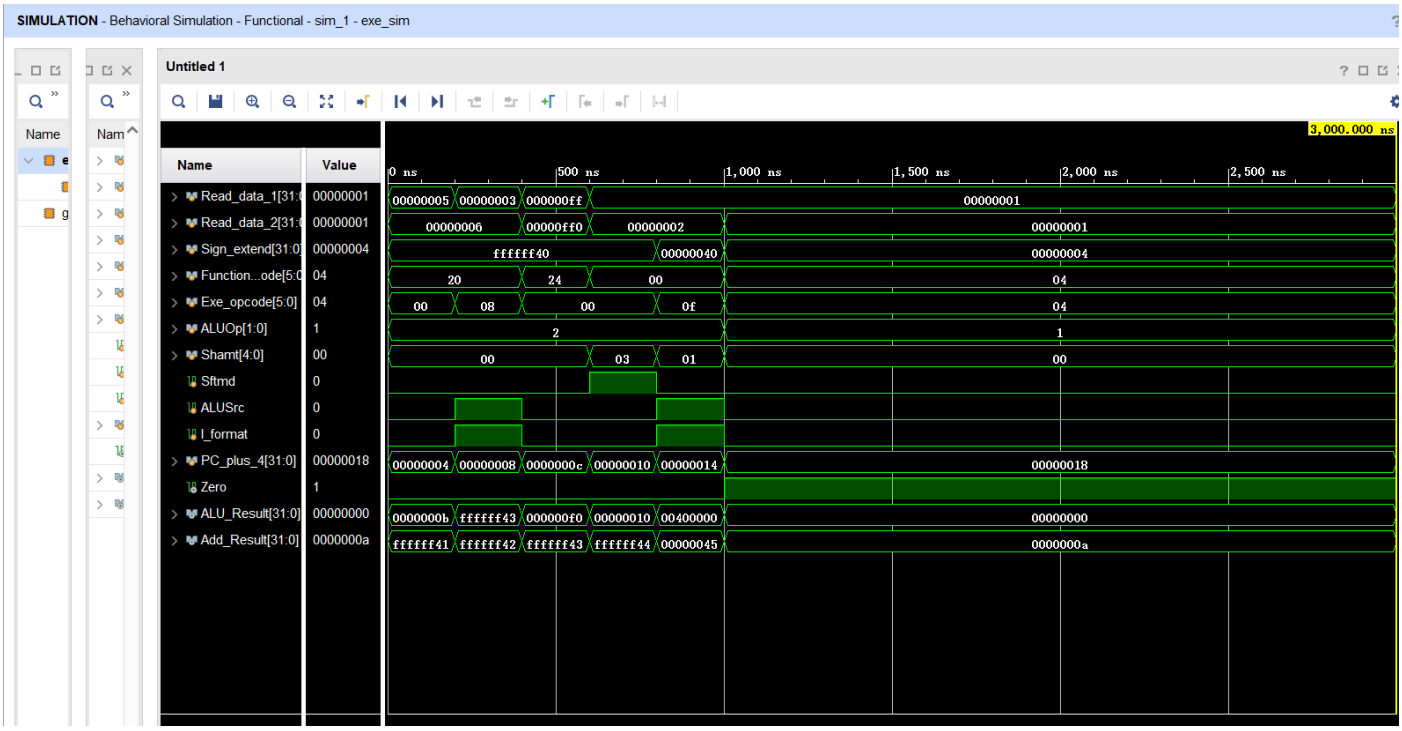
计算机系统综合设计

执行单元仿真的时序

09017227 卓旭（SEU-本 18-09017227-卓旭）

仿真时序图：

exe_sim.v



代码：

executs32.v

```
`include "public.v"

module Executs32(
    input wire[31:0]  Read_data_1,       // r-form rs 从译码单元是 Read_data_1 中来
    input wire[31:0]  Read_data_2,       // r-form rt 从译码单元是 Read_data_2 中来
    input wire[31:0]  Sign_extend,       // i-form 译码单元来的扩展后的立即数
    input wire[5:0]   Function_opcode,  // r-form instructions[5..0] 取指单元来的 R 型的 Func
    input wire[5:0]   Exe_opcode,        // opcode 取值单元来的 Op
    input wire[1:0]   ALUOp,             // 控制单元来的 ALUOp，第一级控制（LW/SW 00，BEQ/BNE
01，R/I 10）
    input wire[4:0]   Shamt,           // 移位量
    input            Sftmd,           // 是否是移位指令
    input             ALUSrc,           // 来自控制单元，表明第二个操作数是立即数（beq、bne 除
外）
    input            I_format,        // 该指令是除了 beq、bne、lw、sw 以外的其他 I 类型指令
    output wire       Zero,            // Zero Flag
    output reg[31:0]  ALU_Result,       // 执行单元的最终运算结果
    output wire[31:0] Add_Result,    // 计算的地址结果
    input wire[31:0]  PC_plus_4         // 来自取指单元的 PC+4
);

    wire[31:0] Ainput, Binput; // ALU 的 A 输入和 B 输入
```

```
    wire signed [31:0] Ainput_signed, Binput_signed;
    reg[31:0] Sinput; // 移位指令的最终结果
    reg[31:0] ALU_output_mux; // ALU 的最终运算结果
    wire[32:0] Branch_Add; // 相对跳转指令目标的运算结果
    wire[2:0] ALU_ctl; // 分级控制用
    wire[5:0] Exe_code; // 分级控制用
    wire[2:0] Sftm;  // 移位指令的类型

    assign Sftm = Function_opcode[2:0];   // 用于判断移位指令的类型
    assign Exe_code = (I_format==0) ? Function_opcode : {3'b000,Exe_opcode[2:0]};
    assign Ainput = Read_data_1; // ALU 的 A 口输入是 rs 的数据
    assign Binput = (ALUSrc == 0) ? Read_data_2 : Sign_extend[31:0]; // ALU 的 B 口输入，可
能是 rt，也可能是立即数
    // 转有符号
    assign Ainput_signed = Ainput;
    assign Binput_signed = Binput;
    // ALU 预算的组合码
    assign ALU_ctl[0] = (Exe_code[0] | Exe_code[3]) & ALUOp[1];
    assign ALU_ctl[1] = ((!Exe_code[2]) | (!ALUOp[1]));
    assign ALU_ctl[2] = (Exe_code[1] & ALUOp[1]) | ALUOp[0];

    always @* begin  // 6 种移位指令的处理
       if (Sftmd)
        case (Sftm[2:0])
            3'b000: begin // Sll rd,rt,shamt  00000
                Sinput = Binput << Shamt;
            end
            3'b010: begin // Srl rd,rt,shamt  00010
                Sinput = Binput >> Shamt;
            end
            3'b100: begin // Sllv rd,rt,rs  000100
                Sinput = Binput << Ainput;
            end
            3'b110: begin // Srlv rd,rt,rs  000110
                Sinput = Binput >> Ainput;
            end
            3'b011: begin // Sra rd,rt,shamt 00011
                Sinput = Binput >>> Shamt;
            end
            3'b111: begin // Srav rd,rt,rs 00111
                Sinput = Binput >>> Ainput;
            end
            default: begin
                Sinput = Binput;
            end
        endcase
       else Sinput = Binput;
    end
```

```
    // 给出最终的运算结果
    always @* begin
        if ( ((ALU_ctl == 3'b111) && (Exe_code[3] == 1)) || ((ALU_ctl[2:1] == 2'b11) &&
(I_format == 1)) ) // 处理 slt 类的问题
            ALU_Result = { 31'd0, ALU_output_mux[31] }; // 符号位为 1 说明 slt 成立
        else if ((ALU_ctl == 3'b101) && (I_format == 1))
            ALU_Result[31:0] = { Binput[15:0], 16'd0 }; // lui，将 B 口输入放到高 16 位
        else if (Sftmd == 1)
            ALU_Result = Sinput; // 移位
        else ALU_Result = ALU_output_mux[31:0]; // 其他情况
    end

    assign Branch_Add = PC_plus_4[31:2] + Sign_extend[31:0]; // 计算相对跳转的目的 PC，这种
写法等价于将指令中的 offset 左移两位
    assign Add_Result = Branch_Add[31:0];
    assign Zero = (ALU_output_mux[31:0] == 32'h00000000) ? `Enable : `Disable; // zero-
flag

    // 根据组合码取不同的值应该做什么运算
    always @(ALU_ctl or Ainput or Binput) begin
        case (ALU_ctl)
            3'b000: begin // and, andi
                ALU_output_mux = Ainput & Binput;
            end
            3'b001: begin // or, ori
                ALU_output_mux = Ainput | Binput;
            end
            3'b010: begin // add, addi. lw, sw
                ALU_output_mux = Ainput_signed + Binput_signed;
            end
            3'b011: begin // addu, addui
                ALU_output_mux = Ainput + Binput;
            end
            3'b100: begin // xor, xori
                ALU_output_mux = Ainput ^ Binput;
            end
            3'b101: begin // nor, lui
                ALU_output_mux = ~(Ainput | Binput);
            end
            3'b110: begin // sub, slti, beq, bne
                ALU_output_mux = Ainput_signed - Binput_signed;
            end
            3'b111: begin // subu, sltiu, slt, sltu
                ALU_output_mux = Ainput - Binput;
            end
            default: begin
                ALU_output_mux = 32'h00000000;
            end
        endcase
```

```
    end
endmodule
```