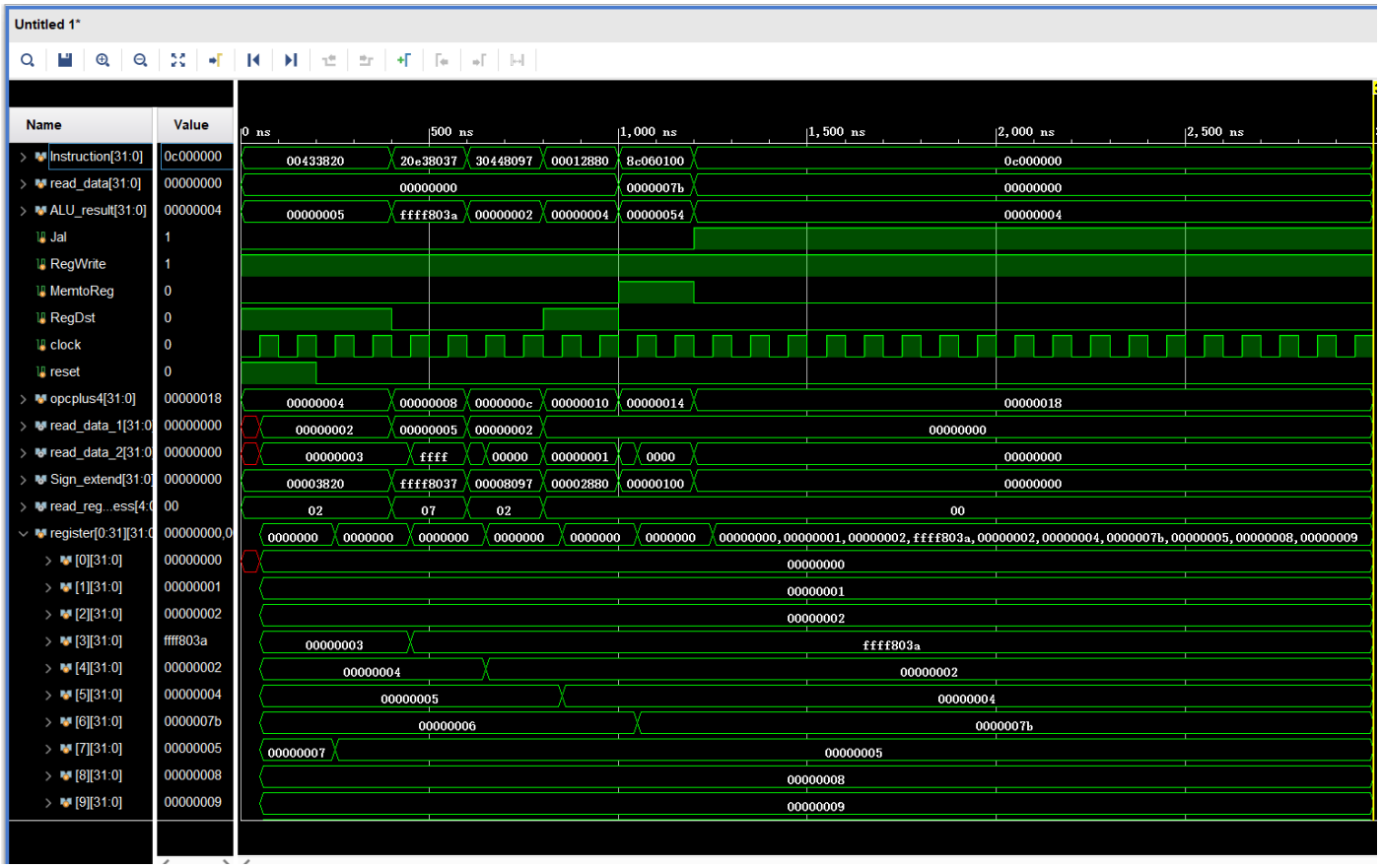
计算机系统综合设计

译码单元仿真的时序

09017227 卓旭（SEU-本 18-09017227-卓旭）

仿真时序图：

idcode32_sim.v



代码：

idecode32.v

```
`include "public.v"

module Idecode32(
    output reg[31:0] read_data_1,  // 读数据 1（rs）
    output reg[31:0] read_data_2,  // 读数据 2（rt）
    input wire[31:0]  Instruction, // 来自取指模块
    input wire[31:0]  read_data,   // 从 DATA RAM or I/O port 取出的数据
    input wire[31:0]  ALU_result,  // 需要扩展立即数到 32 位
    input         Jal,             // 指令是不是 JAL
    input         RegWrite,        // 寄存器写使能
    input         MemorIOtoReg,    // 数据来源是不是 MEM
    input         RegDst,          // 为 1 说明目标寄存器是 rd，否则是 rt
    output reg[31:0] Sign_extend,  // 立即数扩展的结果
    input        clock,
    input         reset,
    input[31:0]  opcplus4,         // 来自取指单元，JAL 中用
    output reg[4:0] read_register_1_address // rs
);
```

```
    reg[31:0] register[0:31];          // 寄存器组共 32 个 32 位寄存器
    reg[4:0] write_register_address;    // 最后决定要写的寄存器
    reg[31:0] write_data;              // 最后决定要写的数据
    reg[4:0] read_register_2_address;  // rt
    reg[4:0] write_register_address_1; // rd r-form
    reg[4:0] write_register_address_0; // rt i-form
    reg[15:0] Instruction_immediate_value; // immediate
    reg[5:0] opcode; // op
    reg sign;

    always @(*) begin
        // 填充上面的一系列参数
        read_register_1_address <= Instruction[`RsRange];
        read_register_2_address <= Instruction[`RtRange];
        read_data_1 = register[read_register_1_address];
        read_data_2 = register[read_register_2_address];

        write_register_address_0 <= Instruction[`RtRange];
        write_register_address_1 <= Instruction[`RdRange];

        Instruction_immediate_value <= Instruction[`ImmedRange];
        opcode <= Instruction[`OpRange];

        sign <= Instruction_immediate_value[15];
        if (opcode == `OP_ANDI || opcode == `OP_ORI || opcode == `OP_XORI || opcode ==
`OP_SLTIU) begin
            // 上面这些指令做零扩展
            Sign_extend <= {16'd0, Instruction_immediate_value[15:0]};
        end else begin
            // 其他做符号位扩展
            Sign_extend <= {{16{sign}}, Instruction_immediate_value[15:0]};
        end
    end

    always @(*) begin   // 这个进程指定不同指令下的目标寄存器
        if (Jal == `Enable) begin
            write_register_address <= 5'b11111; // Jal 指令写$ra（31 号）寄存器
        end else if (RegDst == `Enable) begin // 为 1 说明目标寄存器是 rd，否则是 rt
            write_register_address <= write_register_address_1;
        end else begin
            write_register_address <= write_register_address_0;
        end
    end

    always @(*) begin  // 这个进程基本上是实现结构图中右下的多路选择器,准备要写的数据
        if (Jal == `Enable) begin
            write_data <= opcplus4; // 保存当前 PC
        end else if (MemorIOtoReg == `Disable) begin
            write_data <= ALU_result; // 来源是 ALU 运算结果，而非 MEM
```

```
        end else begin
            write_data <= read_data; // 来源是 MEM
        end
    end

    integer i;
    always @(posedge clock) begin       // 本进程写目标寄存器
        if (reset == `Enable) begin             // 初始化寄存器组
            for (i = 0; i < 32; i = i + 1) register[i] <= i;    // register[0] = 0
        end else if (RegWrite == `Enable) begin  // 注意寄存器 0 恒等于 0
            if (write_register_address != 5'b00000) begin // 不要修改零号寄存器
                register[write_register_address] <= write_data;
            end
        end
    end
endmodule
```