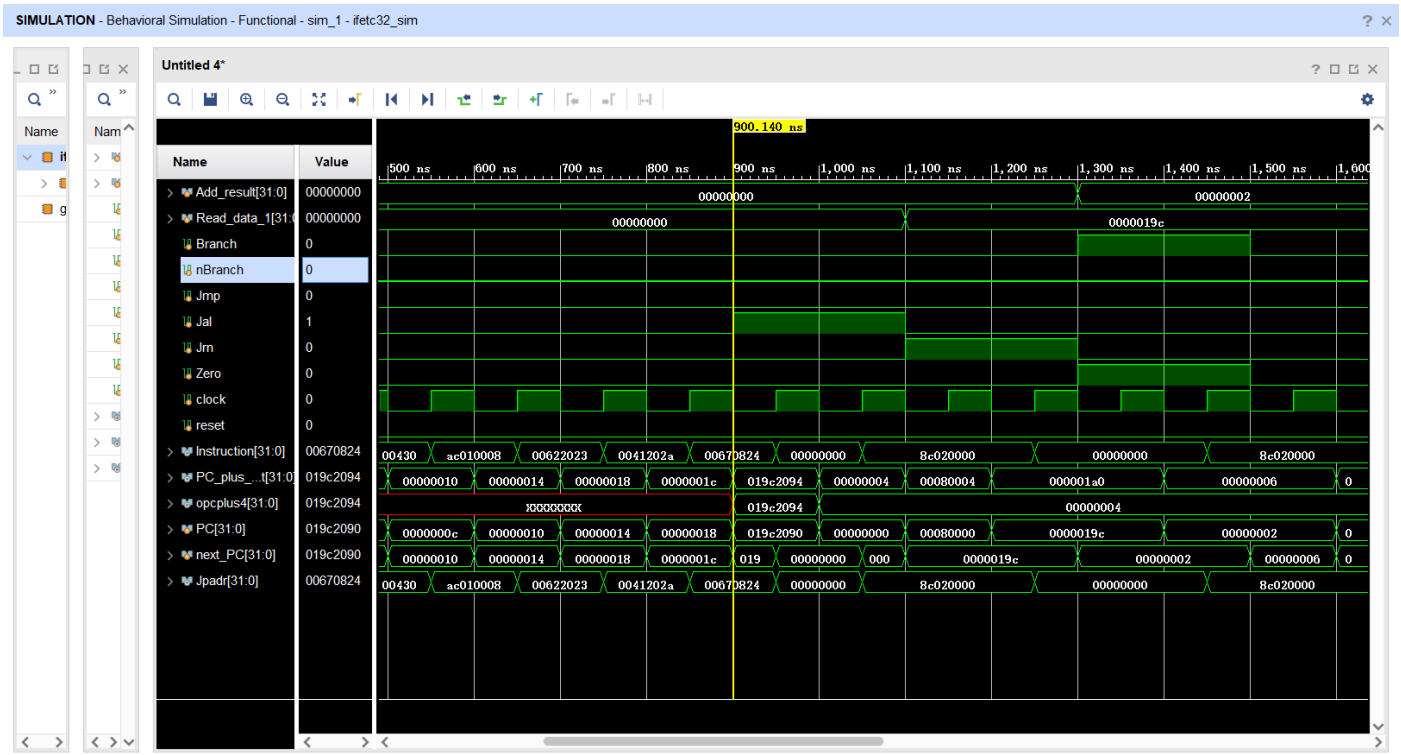
# 计算机系统综合设计

## 取指单元仿真的时序

09017227 卓旭（SEU-本 18-09017227-卓旭）

**仿真时序图：**

***ifetc32_sim.v***




注：由于 MOOC 的演示中指令存储器的 IP 核勾选了 Primitives Output Register，而我所做的实验中没有勾选该选项，因此时序波形图与 MOOC 中将有不同。经验证，如果我亦勾选 Primitives Output Register，则时序图与 MOOC 演示中一致。

**代码：**

***ifetc32.v***

```
`include "public.v"

module Ifetc32(
    output reg[31:0] Instruction,        // 输出指令
    output reg[31:0] PC_plus_4_out,        // PC+4 的结果输出
    input wire[31:0] Add_result,           // 来自执行单元，算出的跳转地址
    input wire[31:0] Read_data_1,          // 来自译码单元，jr 指令用的跳转地址
    input        Branch,                  // beq
    input        nBranch,                 // bne
    input        Jmp,                     // j
    input        Jal,                     // jal
    input        Jrn,                     // jr
    input        Zero,                    // 来自执行单元，1 说明运算结果为 0
    input        clock,                   // clk
    input        reset,                   // rst
```

```
    output reg[31:0] opcplus4              // 返回地址（$31），要早于对 JAL 的修改
);

    reg[31:0] PC_plus_4;
    reg[31:0] PC;
    reg[31:0] next_PC; // 下一条指令的 PC
    wire[31:0] Jpadr;

    // 分配 64KB ROM
    prgrom instmem(
        .clka(clock),         // input wire clka
        .addra(PC[15:2]),     // input wire [13 : 0] addra
        .douta(Jpadr)         // output wire [31 : 0] douta
    );

    always @(*) begin
        PC_plus_4 <= PC + 32'd4;
        PC_plus_4_out <= PC_plus_4;
        Instruction <= Jpadr; // 取出指令
        // beq 且 eq，或 bne 且 ne
        if ((Branch == `Enable && Zero == `Enable) || (nBranch == `Enable && Zero ==
`Disable)) begin
            next_PC <= Add_result;
        // Jr 指令直接覆写 PC
        end else if (Jrn == `Enable) begin
            next_PC <= Read_data_1;
        // 如果是跳转指令，记录返回地址，然后根据指令中的地址字面改写当前 PC
        end else if (Jmp == `Enable || Jal == `Enable) begin
            opcplus4 = PC_plus_4;
            next_PC = { 4'b0000, Jpadr[`AddressRange], 2'b00 };
        // 一般情况下 PC=PC+4
        end else begin
            next_PC <= PC_plus_4;
        end
    end

    always @(negedge clock) begin
        if (reset == `Enable) begin
            PC <= 32'd0;
        end else begin
            PC <= next_PC;
        end
    end

endmodule
```