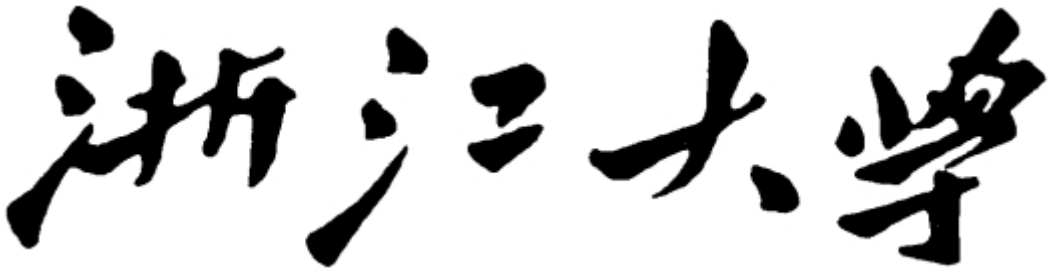


本科实验报告

课程名称:	**计算机组成**
姓 名:	臧可
学 院:	计算机科学与技术学院
系:	计算机科学与技术系
专 业:	计算机科学与技术
学 号:	3180102095
指导教师:	姜晓红

2020年　6月　26日

浙江大学实验报告

课程名称:　计算机组成　　实验类型:　综合

实验项目名称:Lab4--Multi-Cycle-CPU implementation

学生姓名:臧可　　专业:计算机科学 与技术　　学号:3180102095

同组学生姓名:　None　　指导老师:　姜晓红

实验地点:None　　实验日期:　2020年　6月 26日

一、 实验目的和要求

1. **Object:** Build a multi-cycle CPU test application environment

   Requirement:

   - Rebuild Exp03 top-level module with structural description
   - Replace single cycle with multi-cycle CPU core

2. **Object:** Design a 9+ instruction multi-cycle data path and test the data path

   Requirement:

   - Design the logic schematic diagram of the multi-cycle data path
     - R-Type: add, sub, and, or, slt, nor*;
     - I-Type: lw, sw, beq;
     - J-Type: J
   - Design and implement data path with hardware description language
     - ALU and Regs call the modules designed by Exp04
   - Design test plans and test procedures
     - Channel test: I-format channel, R-format channel

3. **Object:** Design a 9+ instruction controller, and then expand the multi-cycle CPU instruction set.
   Set the controller test plan

   Requirement:

   - Design and implement controller with hardware description language
   - Not less than the following instructions
     - R-Type: add, sub, and, or, xor, nor, slt, srl*, jr, jalr, eret *;
     - I-Type: addi, andi, ori, xori, lui, lw, sw, beq, bne, slti
     - J-Type: J, Jal;
   - Encourage personalized design
     - Must be compatible with the basic instruction set
   - OP decoding test: R-format, memory access instruction, branch instruction, transfer instruction
   - Operation control test: Function decoding test

二、 实验内容和原理

Drawing the main ***functions (circuit) graph***, writing out the logical expressions for the control signals; with the key Verilog code segment.

Multi_CPU

```
1  `timescale 1ns / 1ps
2
3  module Multi_CPU(input clk,             //muliti_CPU
4                   input reset,
5                   input MIO_ready,
6                   output[31:0] PC_out,    //TEST
7                   output[31:0] inst_out, //TEST
8                   output mem_w,
```

```
                output[31:0] Addr_out,
                output[31:0] Data_out,
                output[31:0] Data_in,
                output CPU_MIO,
                input INT,
                output [4:0]state     //Test
    );
        wire zero, overflow;
        wire MemRead, MemWrite, IorD, IRWrite, RegWrite, ALUSrcA, PCWrite,
PCWriteCond, Branch;
        wire[1:0] RegDst, MemtoReg, ALUSrcB, PCSource;
        wire[2:0] ALU_operation;
        assign mem_w = MemWrite&&(~MemRead);

        ctrl x_ctrl(.clk(clk),
                    .reset(reset),
                    .Inst_in(inst_out),
                    .zero(zero),
                    .overflow(overflow),
                    .MIO_ready(MIO_ready),
                    .MemRead(MemRead),
                    .MemWrite(MemWrite),
                    .CPU_MIO(CPU_MIO),
                    .IorD(IorD),
                    .IRWrite(IRWrite),
                    .RegWrite(RegWrite),
                    .ALUSrcA(ALUSrcA),
                    .PCWrite(PCWrite),
                    .PCWriteCond(PCWriteCond),
                    .Branch(Branch),
                    .RegDst(RegDst),
                    .MemtoReg(MemtoReg),
                    .ALUSrcB(ALUSrcB),
                    .PCSource(PCSource),
                    .ALU_operation(ALU_operation),
                    .state_out(state_out)
                    );

        M_datapath x_datapath(.clk(clk),
                              .reset(reset),
                              .MIO_ready(MIO_ready),
                              .IorD(IorD),
                              .IRWrite(IRWrite),
                               .RegWrite(RegWrite),
                              .ALUSrcA(ALUSrcA),
                              .PCWrite(PCWrite),
                              .PCWriteCond(PCWriteCond),
                              .Branch(Branch),
                               .RegDst(RegDst),
                              .MemtoReg(MemtoReg),
                              .ALUSrcB(ALUSrcB),
                              .PCSource(PCSource),
                              .ALU_operation(ALU_operation),
                              .data2CPU(Data_in),
                              .M_addr(Addr_out),
                              .zero(zero),
                              .overflow(overflow),
                              .PC_Current(PC_out),
```

```
                                .Inst(inst_out),
                                .data_out(Data_out)
                                );
endmodule
```

M_datapath

```
`timescale 1ns / 1ps

module M_datapath(input clk,
                          input reset,
                          input MIO_ready,
                          input IorD,
                          input IRWrite,
                          input [1:0] RegDst,
                          input RegWrite,
                          input [1:0]MemtoReg,
                          input ALUSrcA,
                          input [1:0]ALUSrcB,
                          input [1:0]PCSource,
                          input PCWrite,
                          input PCWriteCond,
                          input Branch,
                          input [2:0]ALU_operation,
                          output[31:0]PC_Current,
                          input [31:0]data2CPU,
                          output[31:0]Inst,
                          output[31:0]data_out,
                          output[31:0]M_addr,
                          output zero,
                          output overflow
    );
     assign CE = MIO_ready && (PCWrite || (PCWriteCond && Branch && zero));
     assign N0 = 1'b0;
     wire[4:0] rs = Inst[25:21];
     wire[4:0] rt = Inst[20:16];
     wire[4:0] rd = Inst[15:11];
     wire[4:0] Wt_addr;
     wire[15:0] Imm_16 = Inst[15:0];
     wire[31:0] Wt_data, rdata_A, rdata_B, MDR, Alu_A, Alu_B, Imm_32, res, PC_Next, ALU_Out;

     MUX4T1_32 PC_MUX6 (
    .I0(res),
    .I1(ALU_Out),
    .I2({PC_Current[31:28], Inst[25:0], N0, N0}),
    .I3(ALU_Out),
    .s(PCSource[1:0]),
    .o(PC_Next)
    );

      REG32 PC (
    .clk(clk),
    .rst(reset),
    .CE(CE),
    .D(PC_Next),
```

```
49      .Q(PC_Current)
50      );
51
52       Regs regs(
53       .clk(clk),
54       .rst(reset),
55       .L_S(RegWrite),
56       .R_addr_A(rs),
57       .R_addr_B(rt),
58       .Wt_addr(Wt_addr),
59      .wt_data(Wt_data),
60       .rdata_A(rdata_A),
61       .rdata_B(rdata_B)
62       );
63
64       MUX4T1_5 MUX1(
65       .I0(rt),
66       .I1(rd),
67       .I2(5'b11111),
68       .I3(5'b00000),
69       .s(RegDst),
70       .o(Wt_addr)
71       );
72
73       MUX4T1_32 Mem2Reg_MUX2(
74       .I0(ALU_Out),
75       .I1(MDR),
76       .I2({32'h0000_0000}),
77       .I3(32'h0000_0000),
78       .s(MemtoReg),
79       .o(Wt_data)
80       );
81
82       alu x_ALU(
83       .A(Alu_A),
84       .B(Alu_B),
85       .ALU_operation(ALU_operation),
86       .res(res),
87       .zero(zero),
88       .overflow(overflow)
89       );
90
91       MUX2T1_32 A_MUX4 (
92       .I0(rdata_A),
93       .I1(PC_Current),
94       .s(ALUSrcA),
95       .o(Alu_A)
96       );
97
98       MUX4T1_32 B_MUX3 (
99       .I0(rdata_B),
100      .I1(32'd4),
101      .I2(Imm_32),
102      .I3({Imm_32[29:0], N0, N0}),
103      .s(ALUSrcB),
104      .o(Alu_B)
105      );
106
```

```
     REG32 ALUout (
    .clk(clk),
    .rst(1'b0),
    .CE(1'b1),
    .D(res),
    .Q(ALU_Out)
    );

     REG32 IR (
    .clk(clk),
    .rst(reset),
    .CE(IRWrite),
    .D(data2CPU),
    .Q(Inst)
    );

     REG32 reg_MDR (
    .clk(clk),
    .rst(1'b0),
    .CE(1'b1),
    .D(data2CPU),
    .Q(MDR)
    );

     MUX2T1_32 MUX5 (
     .I0(ALU_Out),
     .I1(PC_Current),
     .s(IorD),
     .o(M_addr)
     );

     Ext_32 U5 (
    .imm_16(Imm_16),
    .imm_32(Imm_32)
    );

     assign data_out = rdata_B;

endmodule
```

M_datapath_sim

M_datapath simulation excitation code

```
`timescale 1ns / 1ps

module m_datapath_sim;

    // Inputs
    reg clk;
    reg reset;
    reg MIO_ready;
    reg IorD;
    reg IRWrite;
    reg [1:0] RegDst;
    reg RegWrite;
    reg [1:0] MemtoReg;
```

```
    reg ALUSrcA;
    reg [1:0] ALUSrcB;
    reg [1:0] PCSource;
    reg PCWrite;
    reg PCWriteCond;
    reg Branch;
    reg [2:0] ALU_operation;
    reg [31:0] data2CPU;

    // Outputs
    wire [31:0] PC_Current;
    wire [31:0] Inst;
    wire [31:0] data_out;
    wire [31:0] M_addr;
    wire zero;
    wire overflow;

    // Instantiate the Unit Under Test (UUT)
    M_datapath uut (
        .clk(clk),
        .reset(reset),
        .MIO_ready(MIO_ready),
        .IorD(IorD),
        .IRWrite(IRWrite),
        .RegDst(RegDst),
        .RegWrite(RegWrite),
        .MemtoReg(MemtoReg),
        .ALUSrcA(ALUSrcA),
        .ALUSrcB(ALUSrcB),
        .PCSource(PCSource),
        .PCWrite(PCWrite),
        .PCWriteCond(PCWriteCond),
        .Branch(Branch),
        .ALU_operation(ALU_operation),
        .PC_Current(PC_Current),
        .data2CPU(data2CPU),
        .Inst(Inst),
        .data_out(data_out),
        .M_addr(M_addr),
        .zero(zero),
        .overflow(overflow)
    );

    initial begin
        // Initialize Inputs
        clk = 0;
        reset = 1;
        MIO_ready = 1;
        IorD = 0;
        IRWrite = 1;
        RegDst = 0;
        RegWrite = 0;
        MemtoReg = 0;
        ALUSrcA = 0;
        ALUSrcB = 0;
        PCSource = 0;
        PCWrite = 0;
        PCWriteCond = 0;
```

```
        Branch = 0;
        ALU_operation = 0;
        data2CPU = 0;

        // Wait 100 ns for global reset to finish
        #100;

        // Add stimulus here
            reset = 0;
                //J 1000
                //Start
                IorD = 0;
                IRWrite = 1;
                RegWrite = 0;
                ALUSrcA = 1;    //PC_Current
                ALUSrcB = 2'b01;
                PCSource = 2'b00;
                PCWrite = 1;
                ALU_operation = 3'b010;//add
                data2CPU = 32'b000010_00000000000000000000001000;
                #40;
                //state1
                IRWrite = 0;
                ALUSrcA = 1;
                ALUSrcB = 2'b11;
                PCWrite = 0;
                ALU_operation = 3'b010; //add
                #40;
                //state 9
                PCWrite = 1;
                PCSource = 2'b10;
                #40;
                //220
                //Beq
                //Start
                IorD = 0;
                IRWrite = 1;
                RegWrite = 0;
                ALUSrcA = 1;
                ALUSrcB = 2'b01;
                PCSource = 2'b00;
                PCWrite = 1;
                ALU_operation = 3'b010;//add
                data2CPU = 32'b000100_01011_00000_0000000000000101; //beq
$t3, $zero, 5
                #40;
                //state 1
                IRWrite = 0;
                ALUSrcA = 1;
                ALUSrcB = 2'b11;
                PCWrite = 0;
                ALU_operation = 3'b010;//add
                #40;
                //state 8
                ALUSrcA = 0;//A
                ALUSrcB = 00;//B
                ALU_operation = 3'b110; //sub
                PCWriteCond = 1;
```

```
                Branch = 1;
                PCSource = 2'b01;
                #40;
                PCWriteCond = 0;
                //340
                //Lw
                //Start
                IorD = 0;
                IRWrite = 1;
                RegWrite = 0;
                ALUSrcA = 1;
                ALUSrcB = 2'b01;
                PCSource = 2'b00;
                PCWrite = 1;
                ALU_operation = 3'b010;//add
                data2CPU = 32'b100011_00000_10010_0000000000000010; //lw $s2, 2($zero)
                #40;
                //state 1
                IRWrite = 0;
                ALUSrcA = 1;
                ALUSrcB = 2'b11;
                PCWrite = 0;
                ALU_operation = 3'b010;//add
                #40;
                //state 2
                ALUSrcA = 0;
                ALUSrcB = 10;
                ALU_operation = 3'b010;//add
                #40;
                //state 3
                IorD = 1;
                data2CPU = 32'h5a5a5a5a;
                #40;
                //state 4
                RegDst = 2'b00;
                RegWrite = 1;
                MemtoReg = 2'b01;
                #40;
                //540
                //Add
                //Start
                IorD = 0;
                IRWrite = 1;
                RegWrite = 0;
                ALUSrcA = 1;
                ALUSrcB = 2'b01;
                PCSource = 2'b00;
                PCWrite = 1;
                ALU_operation = 3'b010;//add
                data2CPU = 32'h000000_00000_10010_10001_00000_100000; //add $s1, $zero, $s2
                #40;
                //state1
                IRWrite = 0;
                ALUSrcA = 1;//PC_Current
                ALUSrcB = 2'b11;
                PCWrite = 0;
```

```
                ALU_operation = 3'b010;//add
                #40;
                //state6
                ALUSrcA = 0;
                ALUSrcB = 00;
                ALU_operation = 3'b010; //add
                #40;
                //state7
                RegDst = 2'b01;
                RegWrite = 1;
                MemtoReg = 2'b00;
                #40;
                //700

    end

    always begin
        clk = 1; #10;
        clk = 0; #10;
    end

endmodule
```

ctrl

```
`timescale 1ns / 1ps

module ctrl(input clk,
            input reset,
            input [31:0]Inst_in,
            input zero,
            input overflow,
            input MIO_ready,                //外部输入=1
            output reg MemRead,
            output reg MemWrite,
            output reg[2:0]ALU_operation, //ALU_Control
            output [4:0] state_out,

            output reg CPU_MIO,
            output reg IorD,
            output reg IRWrite,
            output reg [1:0]RegDst,         //预留2位
            output reg RegWrite,
            output reg [1:0]MemtoReg,       //预留2位
            output reg ALUSrcA,
            output reg [1:0]ALUSrcB,
            output reg [1:0]PCSource,
            output reg PCWrite,
            output reg PCWriteCond,
            output reg Branch
    );
    /*状态变量*/
    parameter IF      = 5'b0000, ID     = 5'b00001, EX_R  = 5'b00010,
EX_Mem = 5'b00011, EX_I  = 5'b00100, Lui_WB = 5'b00101;
```

```
    parameter EX_Beq = 5'b00110, EX_Bne = 5'b00111, EX_Jr = 5'b01000,
EX_Jal = 5'b01001, EX_J  = 5'b01010, Mem_RD = 5'b01011;
    parameter Mem_WD = 5'b01100, R_WB   = 5'b01101, I_WB  = 5'b01110,
LW_WB  = 5'b01111, Error = 5'b11111;
    parameter AND = 3'b000, OR = 3'b001, ADD = 3'b010, SUB = 3'b110, NOR =
3'b100, SLT = 3'b111, XOR = 3'b011, SRL = 3'b101;
    parameter value0  = 17'h12821, value1  = 17'h00060, value2  =
17'h00050, value3  = 17'h06001;
    parameter value4  = 17'h00208, value5  = 17'h05001, value6  =
17'h00010, value7  = 17'h0001a;
    parameter value8  = 17'h08090, value9  = 17'h10160, value10 =
17'h00050, value11 = 17'h00058;
    parameter value12 = 17'h00468, value13 = 17'h08090, value14 =
17'h10010, value15 = 17'h1076c;
    /*输出变量宏定义*/
    `define CPU_ctrl_signals {PCWrite, PCWriteCond, IorD, MemRead,
MemWrite, IRWrite, MemtoReg[1:0], PCSource[1:0], ALUSrcB[1:0], ALUSrcA,
RegWrite, RegDst[1:0], CPU_MIO}
    /*状态转换*/
    reg [4:0] state;
    assign state_out = state;
    always@(posedge clk or posedge reset)begin
        if(reset == 1) state <= IF;
        else begin
            case(state)
                IF:         if(MIO_ready) state <= ID;
                            else          state <= IF;
                ID:         begin
                                case(Inst_in[31:26])
                                6'b000000: begin
                                                case (Inst_in[5:0])
                                                    6'b001000: state <=
EX_Jr;
                                                    default:   state <=
EX_R;
                                                endcase
                                            end
                                6'b100011: state <= EX_Mem;   //LW
                                6'b101011: state <= EX_Mem;   //SW
                                6'b001000: state <= EX_I;    //addi
                                6'b001100: state <= EX_I;    //andi
                                6'b001101: state <= EX_I;    //ori
                                6'b001110: state <= EX_I;    //xori
                                6'b001010: state <= EX_I;    //slti
                //              6'b001011: state <= EX_I; //I-type
                                6'b001111: state <= Lui_WB;   //Lui
                                6'b000100: state <= EX_Beq;  //Beq
                                6'b000101: state <= EX_Bne;  //Bne
                                6'b000011: state <= EX_Jal;      //Jal
                                6'b000010: state <= EX_J;        //J
                                default:   state <= Error;
                                endcase
                            end
                EX_Mem:     begin
                                case(Inst_in[31:26])
                                    6'b100011: state <= Mem_RD; //LW
                                    6'b101011: state <= Mem_WD; //SW
                                    default:   state <= Error;
```

```
                                endcase
                            end
                EX_R:       state <= R_WB;
                EX_I:       state <= I_WB;
                EX_Beq:     state <= IF;
                EX_Bne:     state <= IF;
                EX_Jal:     state <= IF;
                EX_Jr:      state <= IF;
                EX_J:          state <= IF;
                Mem_RD:     state <= LW_WB;
                LW_WB:      state <= IF;
                Mem_WD:     state <= IF;
                R_WB:       state <= IF;
                I_WB:          state <= IF;
                Lui_WB:     state <= IF;
                Error:      state <= Error;
                default:       state <= Error;
            endcase
        end
    end

    always@*begin
        case(state)
            IF:       begin `CPU_ctrl_signals <= value0; Branch <= 0; ALU_operation = ADD; end
            ID:       begin `CPU_ctrl_signals <= value1; Branch <= 0; ALU_operation = ADD; end
            EX_Mem: begin `CPU_ctrl_signals <= value2; Branch <= 0; ALU_operation = ADD; end
            Mem_RD: begin `CPU_ctrl_signals <= value3; Branch <= 0; ALU_operation = ADD; end
            LW_WB:  begin `CPU_ctrl_signals <= value4; Branch <= 0; ALU_operation = ADD; end
            Mem_WD: begin `CPU_ctrl_signals <= value5; Branch <= 0; ALU_operation = ADD; end
            EX_R:   begin
                        `CPU_ctrl_signals <= value6;
                        Branch <= 0;
                        case(Inst_in[5:0])
                            6'b100000: ALU_operation = ADD;
                            6'b100010: ALU_operation = SUB;
                            6'b100100: ALU_operation = AND;
                            6'b100101: ALU_operation = OR;
                            6'b100111: ALU_operation = NOR;
                            6'b101010: ALU_operation = SLT;
                            6'b000010: ALU_operation = SRL;
                            6'b000000: ALU_operation = XOR;
                            6'b001000: ALU_operation = ADD; //jr
                            default:   ALU_operation = ADD;
                        endcase
                    end
            R_WB:   begin `CPU_ctrl_signals <= value7 ; Branch <= 0; ALU_operation = ADD; end
            EX_Beq: begin `CPU_ctrl_signals <= value8 ; Branch <= 1; ALU_operation = SUB; end
            EX_J:     begin `CPU_ctrl_signals <= value9 ; Branch <= 0; ALU_operation = ADD; end
```

```
            EX_I:   begin `CPU_ctrl_signals <= value10; Branch <= 0; ALU_operation = SLT; end
            I_WB:   begin `CPU_ctrl_signals <= value11; Branch <= 0; ALU_operation = ADD; end
            Lui_WB: begin `CPU_ctrl_signals <= value12; Branch <= 0; ALU_operation = ADD; end
            EX_Bne: begin `CPU_ctrl_signals <= value13; Branch <= 0; ALU_operation = SUB; end
            EX_Jr:  begin `CPU_ctrl_signals <= value14; Branch <= 0; ALU_operation = ADD; end
            EX_Jal: begin `CPU_ctrl_signals <= value15; Branch <= 0; ALU_operation = ADD; end
            default:begin `CPU_ctrl_signals <= value0 ; Branch <= 0; ALU_operation = ADD; end
        endcase
    end


endmodule

```

ctrl_sim

ctrl simulation excitation code

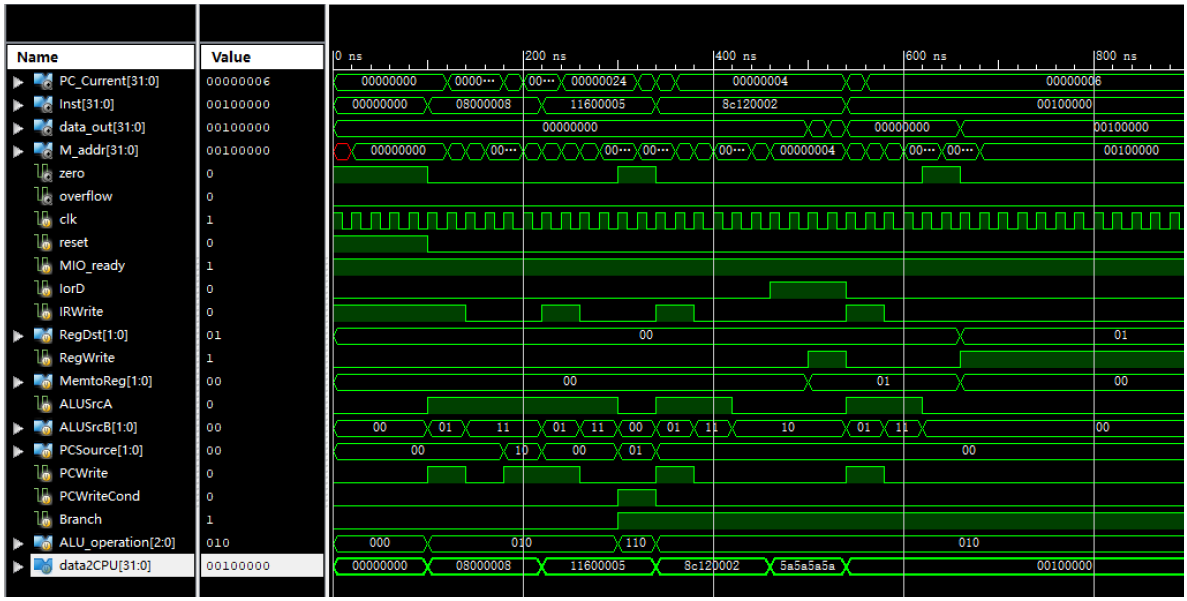
```
`timescale 1ns / 1ps

module ctrl_sim;

    // Inputs
    reg clk;
    reg reset;
    reg [31:0] Inst_in;
    reg zero;
    reg overflow;
    reg MIO_ready;

    // Outputs
    wire [4:0]state_out;
    wire MemRead;
    wire MemWrite;
    wire [2:0] ALU_operation;
    //wire [4:0] state;
    wire CPU_MIO;
    wire IorD;
    wire IRWrite;
    wire [1:0] RegDst;
    wire RegWrite;
    wire [1:0] MemtoReg;
    wire ALUSrcA;
    wire [1:0] ALUSrcB;
    wire [1:0] PCSource;
    wire PCWrite;
    wire PCWriteCond;
    wire Branch;
```

```
31
32      // Instantiate the Unit Under Test (UUT)
33      ctrl uut (
34          .clk(clk),
35          .reset(reset),
36          .Inst_in(Inst_in),
37          .zero(zero),
38          .overflow(overflow),
39          .MIO_ready(MIO_ready),
40          .MemRead(MemRead),
41          .MemWrite(MemWrite),
42          .ALU_operation(ALU_operation),
43          .state_out(state_out),
44          .CPU_MIO(CPU_MIO),
45          .IorD(IorD),
46          .IRWrite(IRWrite),
47          .RegDst(RegDst),
48          .RegWrite(RegWrite),
49          .MemtoReg(MemtoReg),
50          .ALUSrcA(ALUSrcA),
51          .ALUSrcB(ALUSrcB),
52          .PCSource(PCSource),
53          .PCWrite(PCWrite),
54          .PCWriteCond(PCWriteCond),
55          .Branch(Branch)
56      );
57
58      initial begin
59          // Initialize Inputs
60          clk = 0;
61          reset = 1;
62          Inst_in = 0;
63          zero = 0;
64          overflow = 0;
65          MIO_ready = 1;
66
67          #10;
68          reset = 0;
69          // R-type
70          Inst_in = 32'b000000_00000_00000_00000_00000_100000;// add
$s0<=$zero+$zero
71          #80;
72          // I-type
73          Inst_in = 32'b001000_00000_01000_0000000000000001;// addi
$t0<=$zero+1
74          #80;
75          // condition
76          Inst_in = 32'b000100_10110_10111_1111111111101110;// beq $s6 $s7
1111111111101110
77          #60;
78          zero = 1;
79          #60;
80          Inst_in = 32'b000101_10110_10111_1111111111101110;// bne $s6 $s7
1111111111101110
81          #60;
82          zero = 0;
83          #60;
84          // sw
```

```
85          Inst_in = 32'b101011_01001_10001_0000000000010100;//sw
mem[$t1+20]=$s1
86          #80;
87          // lw
88          Inst_in = 32'b100011_01001_10001_0000000000010100;//lw $s1=$t1+20
89          #100
90          // jump
91          Inst_in = 32'b000010_00000000000000000000001000;//j
92          #60;
93          Inst_in = 32'b000011_00000000000000010000000000;//jal
94          #60;
95          Inst_in = 32'b000000_10000_00000_00000_00000_001000;//jr $31
96          #60
97          // lui
98          Inst_in = 32'b001111_00000_10000_0000000000001111;//lui
$8=15*65536
99
100     end
101
102     always begin
103         clk = 1;#10;
104         clk = 0;#10;
105     end
106
107 endmodule
```

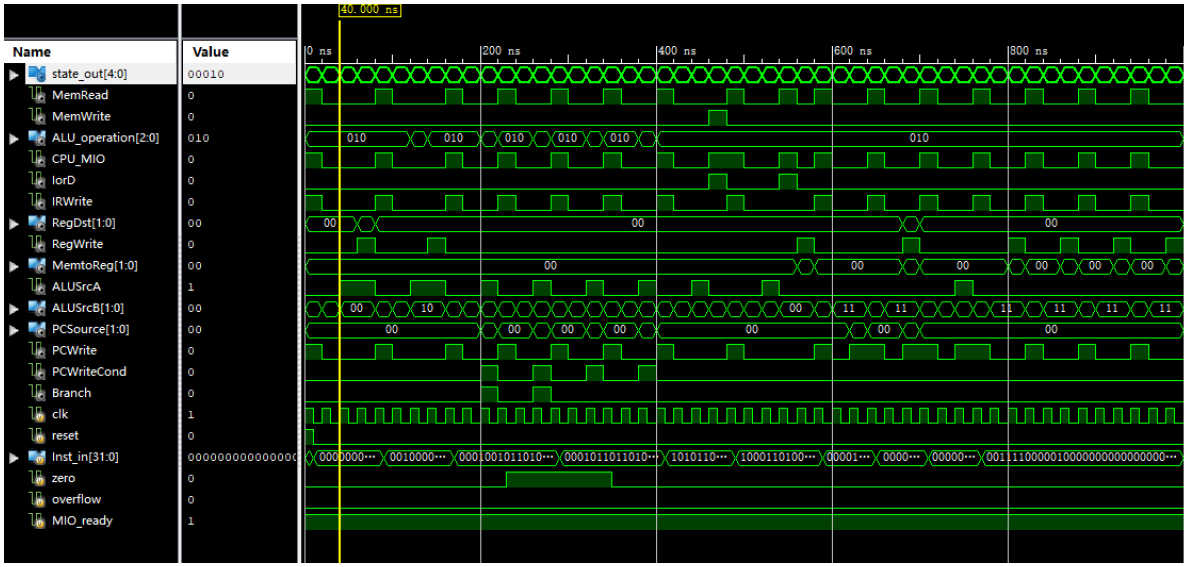
三、 实验过程和数据记录及结果分析

M_datapath



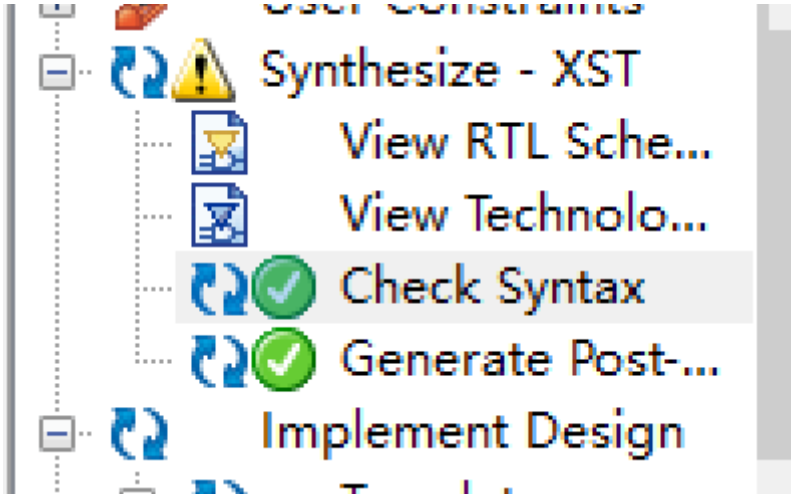
Simulation results are as expected.

ctrl



Simulation results are as expected.

Top



四、 讨论与心得

In this experiment, I have a deeper understanding of the principle of multi-cycle CPU through the simulation of datapath and ctrl modules. During the simulation, the clock cycle must be adjusted to make the multi-cycle CPU simulate the expected result.