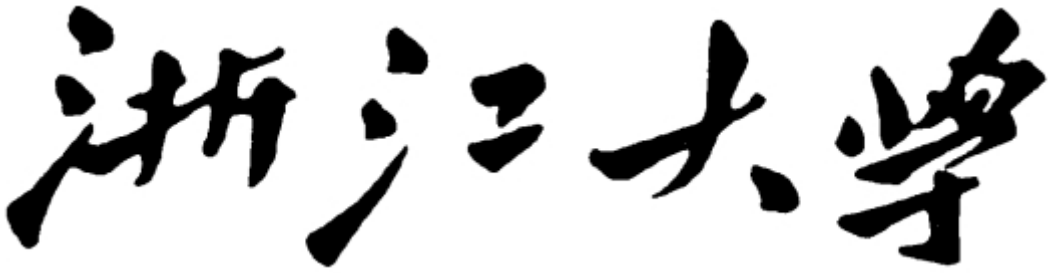


本科实验报告

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 07 月 03 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： Lab5: Microprogrammed Controller

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： 东四509 实验日期： 2020年 07月 03日

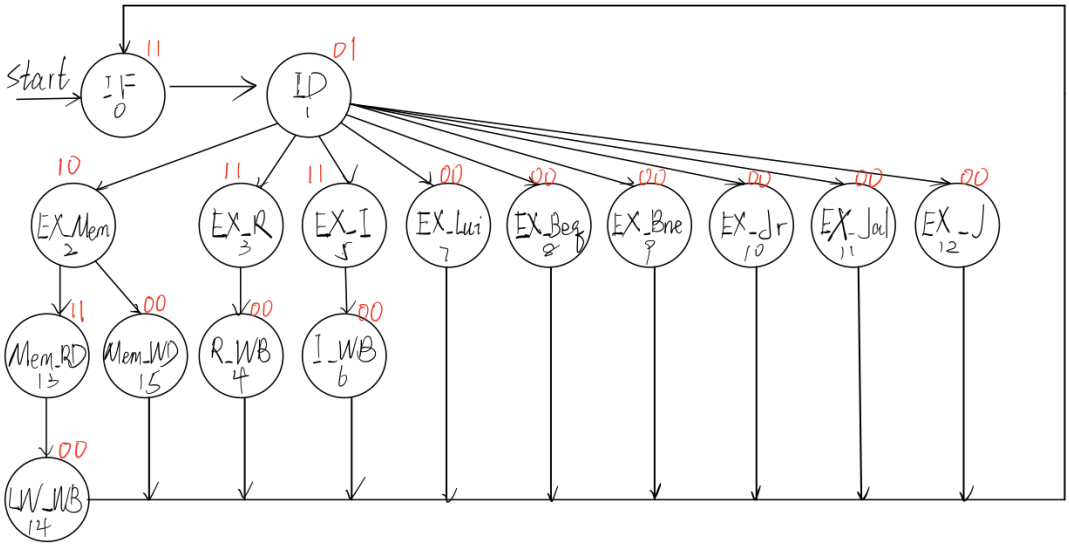
一、 实验目的和要求

Object: implement the Multiple-cycle-CPU controller via microprogramming.

Requirement: Implement and simulate with verilog code.

二、 实验内容和原理

State machine diagram



Microinstruction list

输出控制信号	微指令																
	00000	00001	00010	00011	00100	00101	00110	00111	01000	01001	01010	01011	01100	01101	01110	01111	10000
	IF	ID	EX_Mem	EX_R	R_WB	EX_I	I_WB	EX_Lui	EX_Beq	EX_Bne	EX_Jr	Ex_Jal	EX_J	Mem_RD	LW_WB	Mem_WD	Error
PCWrite	1	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0
PCWriteCond	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0
IorD	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0
MemRead	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
MemWrite	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
IRWrite	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
MemToReg1	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0
MemToReg0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0
PCSource1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0
PCSource0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0
ALUSrcB1	0	1	1	0	0	1	1	1	0	0	0	0	1	1	0	1	0
ALUSrcB0	1	1	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0
ALUSrcA	0	0	1	1	1	1	1	0	1	1	1	0	0	1	0	1	0
RegWrite	0	0	0	0	1	0	1	1	0	0	0	1	0	0	1	0	0
RegDst1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
RegDst0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
CPU_MIO	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0
Branch	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
AddrCtrl1	1	0	1	1	0	1	0	0	0	0	0	0	0	1	0	0	0
Addrctrl0	1	1	0	1	0	1	0	0	0	0	0	0	0	1	0	0	0

AddrCtrl controls the next state, 00 represents returning to Fetch, Dispatch1 represents the first state to complete the branch, Dispatch2 represents the branch at lw/sw, and 11 represents sequential execution

Sequencing	Seq	11
	Fetch	00
	Dispatch1	01
	Dispatch2	10

Verilog code of the microcontroller

```
module Micro_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
    );
    reg [1:0]AddrCtrl;//00-Fetch, 01-Dispatch1, 10-Dispatch2, 11-Seq
    /*20位微指令*/
    `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, Branch, AddrCtrl}
    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 IF   = 5'b00000, ID     = 5'b00001, EX_Mem = 5'b00010, EX_R
  = 5'b00011, R_WB  = 5'b00100, EX_I   = 5'b00101;
    parameter I_WB = 5'b00110, EX_Lui = 5'b00111, EX_Beq = 5'b01000,
EX_Bne = 5'b01001, EX_Jr = 5'b01010, EX_Jal = 5'b01011;
    parameter EX_J = 5'b01100, Mem_RD = 5'b01101, LW_WB  = 5'b01110,
Mem_WD = 5'b01111, Error = 5'b10000;
    parameter value0  = 20'b1001_0100_0001_0000_1011, value1  =
20'b0000_0000_0011_0000_0001;//IF,ID
    parameter value2  = 20'b0000_0000_0010_1000_0010, value3  =
20'b0000_0000_0000_1000_0011;//EX_Mem, EX_R
    parameter value4  = 20'b0000_0000_0000_1101_0000, value5  =
20'b0000_0000_0010_1000_0011;//R_WB, EX_I
    parameter value6  = 20'b0000_0000_0010_1100_0000, value7  =
20'b0000_0010_0011_0100_0000;//I_WB, EX_Lui
```

```
    parameter value8  = 20'b0100_0000_0100_1000_0000, value9  =
20'b0100_0000_0100_1000_0100;//EX_Beq, EX_Bne
    parameter value10 = 20'b1000_0000_0000_1000_0000, value11 =
20'b1000_0011_1011_0110_0000;//EX_Jr, EX_Jal
    parameter value12 = 20'b1000_0000_1011_0000_0000, value13 =
20'b0011_0000_0010_1000_1011;//EX_J, Mem_RD
    parameter value14 = 20'b0000_0001_0000_0100_0000, value15 =
20'b0010_1000_0010_1000_1000;//LW_WB, Mem_WD
    reg [4:0] state, Dispatch1, Dispatch2;
    reg [19:0] MicroMem [16:0];
    wire [4:0] next_state;
    wire [19:0]MicroInst;
    assign state_out = state;

    always@(posedge clk or posedge reset)begin
        if(reset == 1) begin//微指令初始化存入MicroMem
            state = 5'b00000;
            MicroMem[0]  = value0;
            MicroMem[1]  = value1;
            MicroMem[2]  = value2;
            MicroMem[3]  = value3;
            MicroMem[4]  = value4;
            MicroMem[5]  = value5;
            MicroMem[6]  = value6;
            MicroMem[7]  = value7;
            MicroMem[8]  = value8;
            MicroMem[9]  = value9;
            MicroMem[10] = value10;
            MicroMem[11] = value11;
            MicroMem[12] = value12;
            MicroMem[13] = value13;
            MicroMem[14] = value14;
            MicroMem[15] = value15;
            MicroMem[16] = 20'b00000_0000_0000_0000_0000;
        end
        else
            state = next_state;
    end
    assign MicroInst = MicroMem[state];//取指
    //Dispatch
    always@(Inst_in[31:26] or Inst_in[5:0]) begin
        case(Inst_in[31:26])
            6'b000000:  begin
                            case (Inst_in[5:0])
                                6'b001000: Dispatch1 <= EX_Jr;
                                default:   Dispatch1 <= EX_R;
                            endcase
                        end
            6'b100011: begin
                                Dispatch1 <= EX_Mem;  //LW
                                Dispatch2 <= Mem_RD;
                            end
            6'b101011: begin
                                Dispatch1 <= EX_Mem;  //SW
                                Dispatch2 <= Mem_WD;
                        end
            6'b001000: Dispatch1 <= EX_I;    //addi
            6'b001100: Dispatch1 <= EX_I;    //andi
```

```
            6'b001101: Dispatch1 <= EX_I;    //ori
            6'b001110: Dispatch1 <= EX_I;    //xori
            6'b001010: Dispatch1 <= EX_I;    //slti
            6'b001111: Dispatch1 <= EX_Lui;  //Lui
            6'b000100: Dispatch1 <= EX_Beq;  //Beq
            6'b000101: Dispatch1 <= EX_Bne;  //Bne
            6'b000011: Dispatch1 <= EX_Jal;  //Jal
            6'b000010: Dispatch1 <= EX_J;    //J
            default:   Dispatch1 <= Error;
        endcase
    end

    MUX4T1_5 Sequencing(
        .I0(5'b0000),
        .I1(Dispatch1),
        .I2(Dispatch2),
        .I3(state+1'b1),//Seq
        .s(AddrCtrl),
        .o(next_state)
    );

    always@*begin
        `CPU_ctrl_signals = MicroInst;
    end

    always@*begin
        case(state)
            EX_R:
                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
            EX_I:
                case(Inst_in[31:26])
                    6'b001000: ALU_operation = ADD; //addi
                    6'b001100: ALU_operation = AND; //andi
                    6'b001101: ALU_operation = OR;  //ori
                    6'b001110: ALU_operation = XOR; //xori
                    6'b001010: ALU_operation = SLT; //slti
                    default:   ALU_operation = ADD;
                endcase
            EX_Beq:  ALU_operation = SUB;
            EX_Bne:  ALU_operation = SUB;
            default: ALU_operation = ADD;
        endcase
    end

endmodule
```

Microcontroller simulation code

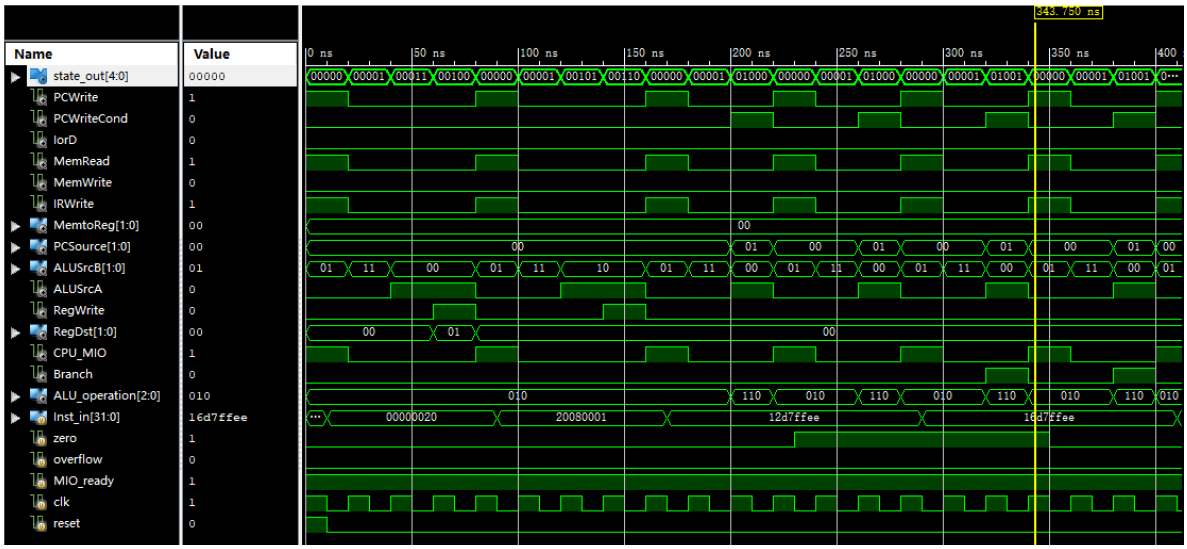
```
module Micro_ctrl_sim;

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

    // Outputs
    wire MemRead;
    wire MemWrite;
    wire [2:0] ALU_operation;
    wire [4:0] state_out;
    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;

    // Instantiate the Unit Under Test (UUT)
    Micro_ctrl uut (
        .clk(clk),
        .reset(reset),
        .Inst_in(Inst_in),
        .zero(zero),
        .overflow(overflow),
        .MIO_ready(MIO_ready),
        .MemRead(MemRead),
        .MemWrite(MemWrite),
        .ALU_operation(ALU_operation),
        .state_out(state_out),
        .CPU_MIO(CPU_MIO),
        .IorD(IorD),
        .IRWrite(IRWrite),
        .RegDst(RegDst),
        .RegWrite(RegWrite),
        .MemtoReg(MemtoReg),
        .ALUSrcA(ALUSrcA),
        .ALUSrcB(ALUSrcB),
        .PCSource(PCSource),
        .PCWrite(PCWrite),
        .PCWriteCond(PCWriteCond),
        .Branch(Branch)
    );

    initial begin
```

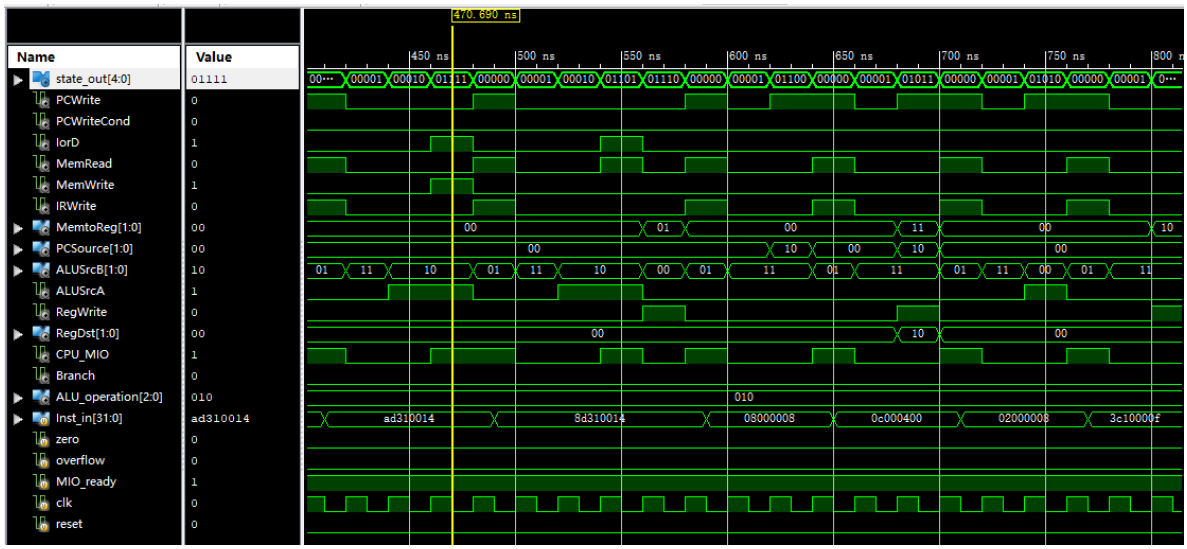
```
            // Initialize Inputs
        clk = 0;
        reset = 1;
        Inst_in = 0;
        zero = 0;
        overflow = 0;
        MIO_ready = 1;

        #10;
        reset = 0;
        // R-type
        Inst_in = 32'b000000_00000_00000_00000_00000_100000;// add $s0<=$zero+$zero
        #80;
        // I-type
        Inst_in = 32'b001000_00000_01000_0000000000000001;// addi $t0<=$zero+1
        #80;
        // condition
        Inst_in = 32'b000100_10110_10111_1111111111101110;// beq $s6 $s7 1111111111101110
        #60;
        zero = 1;
        #60;
        Inst_in = 32'b000101_10110_10111_1111111111101110;// bne $s6 $s7 1111111111101110
        #60;
        zero = 0;
        #60;
        // sw
        Inst_in = 32'b101011_01001_10001_0000000000010100;//sw mem[$t1+20]=$s1
        #80;
        // lw
        Inst_in = 32'b100011_01001_10001_0000000000010100;//lw $s1=$t1+20
        #100
        // jump
        Inst_in = 32'b000010_00000000000000000000001000;//j
        #60;
        Inst_in = 32'b000011_00000000000000010000000000;//jal
        #60;
        Inst_in = 32'b000000_10000_00000_00000_00000_001000;//jr $31
        #60
        // lui
        Inst_in = 32'b001111_00000_10000_0000000000001111;//lui $8=15*65536

    end

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

endmodule
```

It is the same as the original multi-cycle CPU controller simulation excitation code, in order to facilitate the verification of correctness

三、 实验过程和数据记录及结果分析

Micro_ctrl simulation results





Match with the output in the micro-instruction table, correct

四、 讨论与心得

Through this experiment, I have a deeper understanding of the multi-cycle CPU controller.

When writing microinstructions, draw the state machine diagram and microinstruction list is conducive to the writing of verilog code.
It should be noted that the initial value of state is assigned to 0 during initialization, otherwise the simulation result will be x