暨南大学本科实验报告专用纸

课程名称 计算机组成原理 成绩评定

实验项目名称 基于 RISC-V 指令集的单周期 CPU 指导教师 王传胜

实验项目编号 0806006405 实验项目类型 设计 实验地点 N126

学生姓名 甄洛生 学号 2018054625

学院 信息科学技术 系 计算机科学 专业计算机科学与技术

实验时间 2020 年 10 月 27 日 下午～10 月 27 日 午 温度 ℃湿度

一、 实验目的

- 掌握指令执行过程的 5 个阶段
- 掌握每条指令的数据通路选择
- 掌握译码器和控制器的功能和实现
- 掌握数据输入输出处理的方法
- 实现 risc-v 中 RV32I 指令的单周期 CPU
- 利用实现的 risc-v CPU 实现斐波那契数

二、 实验内容

1. 实现 risc-v 中 37 条 RV32I 指令的单周期 cpu
2. 在该 cpu 上实现斐波那契数

三、 实验程序

TOP.v：顶层模块

```
1.  `timescale 1ns / 1ps
2.
3.  module TOP(
4.      input clk,
5.      input rst,
6.      input [5:0] n,
7.      output [6:0] a2g_l,
8.      output [3:0] an_l,
9.      output [6:0] a2g_r,
10.     output [3:0] an_r
11. );
12.     // PC 模块 in/out 寄存器/线网
13.     reg [31:0] next_addr;
14.     wire [31:0] pc_addr;
15.     // ROM 模块 in/out 寄存器/线网
```

```
16.     wire [31:0] ins;
17.     wire [6:0] opcode;
18.     wire [4:0] rs1, rs2, rd;
19.     wire [11:0] imm12;
20.     wire [19:0] imm20;
21.     wire [2:0] funct3;
22.     wire funct7;
23.     // CU 模块 in/out 寄存器/线网
24.     wire cond;
25.     wire reg_we, mem_we, use_imm, use_imm20, from_pc,
26.          from_mem, from_pc4;
27.     wire [1:0] pc_next_select;
28.     wire [2:0] mem_op;
29.     wire [3:0] alu_op;
30.     // RegStack 模块 in/out 寄存器/线网
31.     reg [31:0] reg_wdata;
32.     wire [31:0] reg_data1, reg_data2;
33.     // ALU 模块 in/out 寄存器/线网
34.     wire [31:0] a_in, b_in;
35.     wire [31:0] alu_f;
36.     // RAM 模块 out 寄存器/线网
37.     wire [31:0] mem_data;
38.     // Screen 模块 out 寄存器/线网
39.     wire [31:0] screen_data;
40.
41.     // 下一次 PC 取值的四路选择器
42.     always @(*) begin
43.         case(pc_next_select)
44.             2'b00: next_addr = pc_addr + 32'h4;
45.             2'b01: next_addr = pc_addr + ({{20{imm12[11]}}, imm12[11:0]} <<
    1);
46.             2'b10: next_addr = pc_addr + ({{12{imm20[19]}}, imm20[19:0]} <<
    1);
47.             2'b11: next_addr = alu_f & -32'd2;
48.             default: next_addr = pc_addr + 32'h4;
49.         endcase
50.     end
51.
52.     PC TOP_PC(
53.         .clk(clk),
54.         .rst(rst),
55.         .next(next_addr),
```

```
56.         .addr(pc_addr)
57.     );
58.
59.
60.     ROM TOP_ROM(
61.         .addr(pc_addr),
62.         .ins(ins)
63.     );
64.
65.     ID TOP_ID(
66.         .ins(ins),
67.         .opcode(opcode),
68.         .rs1(rs1),
69.         .rs2(rs2),
70.         .rd(rd),
71.         .imm12(imm12),
72.         .imm20(imm20),
73.         .funct3(funct3),
74.         .funct7(funct7)
75.     );
76.
77.
78.     CU TOP_CU(
79.         .opcode(opcode),
80.         .funct3(funct3),
81.         .funct7(funct7),
82.         .cond(cond),
83.         .reg_we(reg_we),
84.         .mem_we(mem_we),
85.         .mem_op(mem_op),
86.         .alu_op(alu_op),
87.         .use_imm(use_imm),
88.         .use_imm20(use_imm20),
89.         .from_pc(from_pc),
90.         .from_mem(from_mem),
91.         .from_pc4(from_pc4),
92.         .pc_next_select(pc_next_select)
93.     );
94.
95.     // wdata 来源: PC+4 mem alu
96.     always @(*) begin
```

```
97.         if(from_pc4)
98.             reg_wdata = pc_addr + 4;
99.         else if(from_mem)
100.             reg_wdata = mem_data;
101.         else
102.             reg_wdata = alu_f;
103.     end
104.
105.     RegStack TOP_RegStack(
106.         .clk(clk),
107.         .we(reg_we),
108.         .raddr1(rs1),
109.         .rdata1(reg_data1),
110.         .raddr2(rs2),
111.         .rdata2(reg_data2),
112.         .waddr(rd),
113.         .wdata(reg_wdata)
114.     );
115.
116.     // a 端口数据来源: PC rs1
117.     // b 端口数据来源: rs2 imm12 imm20
118.     assign a_in = from_pc ? pc_addr : reg_data1;
119.     assign b_in = use_imm ? (use_imm20 ? {{12{imm20[19]}}, imm20[19:0]} : {
    {20{imm12[11]}}, imm12[11:0]}) : reg_data2;
120.     ALU TOP_ALU(
121.         .a(a_in),
122.         .b(b_in),
123.         .op(alu_op),
124.         .f(alu_f),
125.         .ZR(cond)
126.     );
127.
128.
129.     IOMapping TOP_IOMapping(
130.         .clk(clk),
131.         .we(mem_we),
132.         .mm(mem_op),
133.         .addr(alu_f[7:0]),
134.         .rdata(mem_data),
135.         .wdata(reg_data2),
136.         .switch(n),
137.         .result(screen_data)
```

```
    );

    Screen TOP_Screen(
        .clk(clk),
        .data(screen_data),
        .a2g_l(a2g_l),
        .an_l(an_l),
        .a2g_r(a2g_r),
        .an_r(an_r)
    );

    // 用来作分割线的
    always @(posedge clk)
        $display("                $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$");
endmodule
```

ALU.v：算数逻辑单元模块

```
`timescale 1ns / 1ps

module ALU(
    input [31:0] a,
    input [31:0] b,
    input [3:0] op,
    output reg [31:0] f,
    output ZR               // 零标识位
);
    parameter ZERO = 4'h0, ADD = 4'h1, SUB = 4'h2,
              AND = 4'h3, OR = 4'h4, XOR = 4'h5,
              SLL = 4'h6, SRL = 4'h7, SRA = 4'h8,
              NE = 4'h9, LT = 4'hA, LTU = 4'hB,
              GE = 4'hC, GEU = 4'hD, SL12 = 4'hE,
              SL12ADD = 4'hF;

    always @(*)
    begin
        case(op)
            ZERO:   f = 32'b0;
            ADD:    f = a + b;
            SUB:    f = a - b;
            AND:    f = a & b;
```

```
            OR:     f = a | b;
            XOR:    f = a ^ b;
            SLL:    f = a << b[4:0];
            SRL:    f = a >> b[4:0];
            SRA:    f = $signed(a) >>> b[4:0];
            NE:     if(a != b) f = 32'h0; else f = 32'h1;
            LT:     if($signed(a) < $signed(b)) f = 32'h1; else f = 32'h0;
            LTU:    if(a < b) f = 32'h1; else f = 32'h0;
            GE:     if($signed(a) >= $signed(b)) f = 32'h1; else f = 32'h0;
            GEU:    if(a >= b) f = 32'h1; else f = 32'h0;
            SL12:    f = b << 12;
            SL12ADD: f = (b << 12) + a;
            default: f = 32'b0;
        endcase
    end

    assign ZR = ~(|f);

    initial begin
        $monitor($time,, "ALU->a => 0x%8h\t->b => 0x%8h\t->op => %4b", a, b, op);
        $monitor($time,, "ALU->f => 0x%8h", f);
    end
endmodule
```

ID.v：译码模块

```
`timescale 1ns / 1ps
/*
 * 指令译码器 ID：将 ins 根据 RISC-V 指令集规则
 * 分解成各个有用的字段，提供给 CU 模块。
 */
module ID(
    input [31:0] ins,       // 指令代码
    output [4:0] rs1,       // 原操作数寄存器 1
    output [4:0] rs2,       // 原操作数寄存器 2
    output [4:0] rd,        // 目的操作数寄存器
    output [6:0] opcode,    // 操作代码
    output [11:0] imm12,    // 12 位立即数
    output [19:0] imm20,    // 20 位立即数
```

```
    output [2:0] funct3,    // 3 位 opcode 附加码
    output funct7           // 7 位 opcode 附加码 (这里 1 位的原因稍后说明)
);
    assign opcode = ins[6:0];
    assign rs1 = ins[19:15];
    assign rs2 = ins[24:20];
    assign rd = ins[11:7];
    assign funct3 = ins[14:12];
    /*
     * 虽然 funct7 为 7 位，但实际上我们要实现的
     * 37 条指令里的 funct7 只有 funct7[6] 不同
     */
    assign funct7 = ins[30];

    /*
     * 立即数的译码稍微复杂些，因为他的位置在 ins 里变动
     * 归纳得出五种不同的 imm 分布格式
     */
    wire fm1, fm2, fm3, fm4, fm5;

    // 格式 1 所有 I 类指令和 jalr opcode: 00x0011 和 1100111
    assign fm1 = (~ins[6] & ~ins[5] & ~ins[3] & ~ins[2] & ins[1] & ins[0]) |
                 ((&ins[6:5]) & ~(|ins[4:3]) & (&ins[2:0]));
    // 格式 2 所有 B 类指令 opcode: 1100011
    assign fm2 = (&ins[6:5]) & ~(|ins[4:2]) & (&ins[1:0]);
    // 格式 3 所有 S 类指令 opcode:0100011
    assign fm3 = ~ins[6] & ins[5] & ~(|ins[4:2]) & (&ins[1:0]);
    // 格式 4 jal opcode: 1101111
    assign fm4 = (&ins[6:5]) & ~ins[4] & (&ins[3:0]);
    // 格式 5 所有 U 类指令 opcode: 0x10111
    assign fm5 = ~ins[6] & ins[4] & ~ins[3] & (&ins[2:0]);

    assign imm12 = ({12{fm1}} & ins[31:20]) |
                   ({12{fm2}} & {ins[31], ins[7], ins[30:25], ins[11:8]}) |
                   ({12{fm3}} & {ins[31:25], ins[11:7]});

    assign imm20 = ({20{fm5}} & ins[31:12]) |
                   ({20{fm4}} & {ins[31], ins[19:12], ins[20], ins[30:21]});

endmodule
```

PC.v：程序计数器模块

```
`timescale 1ns / 1ps

// 支持送数的程序计数器，上升沿触发，复位高电平有效
module PC(
    input clk,
    input rst,
    input [31:0] next,
    output reg [31:0] addr
);
    always @(posedge clk)
    begin
        if(rst)
            addr <= 32'b0;
        else
            addr <= next;
    end

    initial
        $monitor($time, , "PC->next => 0x%8h", next);
endmodule
```

ROM.v：只读存储器模块

```
`timescale 1ns / 1ps

module ROM(
    input [31:0] addr,
    output reg [31:0] ins
);
    // 综合前请设置为斐波那契程序
    always @(*)
    begin
        case (addr[31:2])
            // R-type 测试程序
             /*
            30'h0: ins = 32'hffe00093; // addi x1, x0, -2
            30'h1: ins = 32'h00600113; // addi x2, x0, 6
            30'h2: ins = 32'h002081b3; // add x3, x1, x2
            30'h3: ins = 32'h40208233; // sub x4, x1, x2
            30'h4: ins = 32'h0020f2b3; // and x5, x1, x2
```

```
18.             30'h5: ins = 32'h0020e333; // or  x6, x1, x2
19.             30'h6: ins = 32'h0020c3b3; // xor x7, x1, x2
20.             30'h7: ins = 32'h00209433; // sll x8, x1, x2
21.             30'h8: ins = 32'h0020d4b3; // srl x9, x1, x2
22.             30'h9: ins = 32'h4020d533; // sra x10, x1, x2
23.             30'ha: ins = 32'h0020a5b3; // slt x11, x1, x2
24.             30'hb: ins = 32'h00112633; // slt x12, x2, x1
25.             30'hc: ins = 32'h001136b3; // sltu x13, x2, x1
26.             30'hd: ins = 32'h0020b733; // sltu x14, x1, x2
27.              */
28.
29.             // I-type 测试程序
30.              /*
31.             30'h0: ins = 32'h123450b7; // lui x1, 0x12345
32.             30'h1: ins = 32'h7890e093; // ori x1, x1, 0x789
33.             30'h2: ins = 32'hffb00113; // addi x2, x0, -5
34.             30'h3: ins = 32'h00104193; // xori x3, x0, 1
35.             30'h4: ins = 32'h0090f213; // andi x4, x1, 9
36.             30'h5: ins = 32'h00109293; // slli x5, x1, 1
37.             30'h6: ins = 32'h0010d313; // srli x6, x1, 1
38.             30'h7: ins = 32'h40115393; // srai x7, x2, 1
39.             30'h8: ins = 32'h00312413; // slti x8, x2, 3
40.             30'h9: ins = 32'h0090a493; // slti x9, x1, 9
41.             30'ha: ins = 32'h00313b13; // sltiu x22, x2, 3
42.             30'hb: ins = 32'h0051b593; // sltiu x11, x3, 5
43.             30'hc: ins = 32'h00100503; // lb   x10, 1(x0)
44.             30'hd: ins = 32'h00201583; // lh   x11, 2(x0)
45.             30'he: ins = 32'h00402603; // lw   x12, 4(x0)
46.             30'hf: ins = 32'h00804683; // lbu  x13, 8(x0)
47.             30'h10: ins = 32'h01005703; // lhu  x14, 16(x0)
48.              */
49.
50.             // S-type 测试程序
51.              /*
52.             30'h0: ins = 32'h124380b7; // lui x1, 0x12438
53.             30'h1: ins = 32'h7ab0e093; // ori x1, x1, 0x7ab
54.             30'h2: ins = 32'h00100023; // sb x1,0(x0)
55.             30'h3: ins = 32'h001010a3; // sh x1,1(x0)
56.             30'h4: ins = 32'h00102123; // sw x1,2(x0)
57.              */
58.
59.              /* B-type 测试程序
```

```
60.             30'h0: ins = 32'h00100093; // addi x1, x0, 1
61.             30'h1: ins = 32'h7e100fe3; // beq x0, x1, 0xfff
62.             30'h2: ins = 32'h00000463; // beq x0, x0, 8
63.             30'h3: ins = 32'h00000033; // add x0, x0, x0
64.             30'h4: ins = 32'h7e001fe3; // bne x0, x0, 0xfff
65.             30'h5: ins = 32'h00101463; // bne x0, x1, 8
66.             30'h6: ins = 32'h00000033; // add x0, x0, x0
67.             30'h7: ins = 32'h7e00cfe3; // blt x1, x0, 0xfff
68.             30'h8: ins = 32'h7e00efe3; // bltu x1, x0, 0xfff
69.             30'h9: ins = 32'h7e105fe3; // bge x0, x1, 0xfff
70.             30'ha: ins = 32'h7e107fe3; // bgeu x0, x1, 0xfff
71.              */
72.
73.             // U-type 测试程序
74.              /*
75.             30'h0: ins = 32'h123450b7; // lui x1, 0x12345
76.             30'h1: ins = 32'h10000117; // auipc x2, 0x10000
77.              */
78.
79.             // J-type 测试程序
80.              /*
81.             30'h0: ins = 32'h0100006f; // jal x0, 0x10
82.             30'h1: ins = 32'h001000b3; // add x1, x0, x1
83.             30'h2: ins = 32'h001000b3; // add x1, x0, x1
84.             30'h3: ins = 32'h001000b3; // add x1, x0, x1
85.             30'h4: ins = 32'h01000067; // jalr x0, x0, 0x10
86.             30'h5: ins = 32'h001000b3; // add x1, x0, x1
87.             30'h6: ins = 32'h001000b3; // add x1, x0, x1
88.             30'h7: ins = 32'h001000b3; // add x1, x0, x1
89.              */
90.
91.             // 计算斐波那契数程序: 由 Jupiter 汇编实验一得来
92.             // /*
93.             30'h0: ins = 32'h04002503; // lw a0, 0x40(x0) 外设 switch 读 n
94.             30'h1: ins = 32'h00100493; // addi s1, zero, 1
95.             30'h2: ins = 32'h00100913; // addi s2, zero, 1
96.             30'h3: ins = 32'h00300993; // addi s3, zero, 3
97.             30'h4: ins = 32'h00200a13; // addi s4, zero, 2
98.             30'h5: ins = 32'h01355463; // bge a0, s3, fibo
99.             30'h6: ins = 32'h0180006f; // jal zero, output
100.             30'h7: ins = 32'h009002b3; // fibo: add t0, zero, s1
101.             30'h8: ins = 32'h012004b3; // add s1, zero, s2
```

```
            30'h9: ins = 32'h01228933; // add s2, t0, s2
            30'ha: ins = 32'hfff50513; // addi a0, a0, -1
            30'hb: ins = 32'hff4518e3; // bne a0, s4, fibo
            30'hc: ins = 32'h09202023; // output: sw s2, 0x80(x0) 输出至
Screen
            // */
            default: ins = 32'h0;
        endcase
    end

    initial begin
        $monitor($time, , "ROM->addr => 0x%8h", addr);
        $monitor($time, , "ROM->ins  => 0x%8h", ins);
    end
endmodule
```

Screen.v：屏幕模块

```
`timescale 1ns / 1ps

// 基于七段数码管显示数字的屏幕模块
module Screen(
    input clk,
    input [31:0] data,
    output [6:0] a2g_l,
    output [3:0] an_l,
    output [6:0] a2g_r,
    output [3:0] an_r
    );
    wire clk_3hz;
    // 分频器 100MHz -> 3Hz
    divclk Screen_divclk(
        .clk(clk),
        .new_clk(clk_3hz)
    );
    // 左边四个数码管
    digit Screen_left_digits(
        .data(data[31:16]),
        .clk(clk_3hz),
        .a2g(a2g_l),
        .an(an_l)
    );
```

```
    // 右边四个数码管
    digit Screen_right_digits(
        .data(data[15:0]),
        .clk(clk_3hz),
        .a2g(a2g_r),
        .an(an_r)
    );

    initial
        $monitor($time,, "Screen->data => 0x%8h", data);
endmodule
```

CU.v ： 控制单元模块

```
`timescale 1ns / 1ps

/*
 *  根据译码结果生成各个部件的控制信号
 */
module CU(
    input [6:0] opcode,             // RISCV 指令的 opcode 字段
    input [2:0] funct3,             // RISCV 指令的 funct3 字段
    input funct7,                   // RISCV 指令的 funct7[6]比特
    input cond,                     // ALU 的零标志位 ZR
    output reg_we,                  // 寄存器堆写使能
    output mem_we,                  // IOMapping 写使能
    output [2:0] mem_op,            // RAM 操作码
    output [3:0] alu_op,            // ALU 操作码
    output use_imm,                 // ALU 的 a 端口使用立即数
    output use_imm20,               // ALU 的立即数使用 20 位
    output from_pc,                 // ALU 的 a 端口使用 PC 值
    output from_mem,                // 寄存器写数据来自内存
    output from_pc4,                // 寄存器写数据来自 PC+4
    output [1:0] pc_next_select     // 下一 PC 值选择
);
    // 6 种指令类型线网
    wire R_type, I_type, U_type, B_type, S_type, J_type;
    // funct3 信号线网
    wire f3_0, f3_1, f3_2, f3_3, f3_4, f3_5, f3_6, f3_7;
    // 37 条指令信号线网
    wire _add, _sub, _and, _or, _xor, _sll, _srl, _sra,
        _slt, _sltu, _addi, _andi, _ori, _xori, _slti,
```

```
29.         _sltiu, _slli, _srli, _srai, _lw, _lb, _lbu,
30.         _lh, _lhu, _sw, _sb, _sh, _lui, _auipc, _jal,
31.         _jalr, _beq, _bne, _blt, _bltu, _bge, _bgeu;
32.
33.     assign R_type = ~opcode[6] & (&opcode[5:4]) & ~(|opcode[3:2]) & (&opcode
    [1:0]);
34.     assign I_type = ~(|opcode[6:5]) & ~(|opcode[3:2]) & (&opcode[1:0]);
35.     assign U_type = ~opcode[6] & opcode[4] & ~opcode[3] & (&opcode[2:0]);
36.     assign B_type = (&opcode[6:5]) & ~(|opcode[4:2]) & (&opcode[1:0]);
37.     assign S_type = ~opcode[6] & opcode[5] & ~(|opcode[4:2]) & (&opcode[1:0]
    );
38.     assign J_type = (&opcode[6:5]) & ~opcode[4] & (&opcode[2:0]);
39.
40.     assign f3_0 = ~(|funct3[2:0]);
41.     assign f3_1 = ~(|funct3[2:1]) & funct3[0];
42.     assign f3_2 = ~funct3[2] & funct3[1] & ~funct3[0];
43.     assign f3_3 = ~funct3[2] & (&funct3[1:0]);
44.     assign f3_4 = funct3[2] & ~(|funct3[1:0]);
45.     assign f3_5 = funct3[2] & ~funct3[1] & funct3[0];
46.     assign f3_6 = (&funct3[2:1]) & ~funct3[0];
47.     assign f3_7 = (&funct3[2:0]);
48.
49.     assign _add = R_type & f3_0 & ~funct7;
50.     assign _sub = R_type & f3_0 &  funct7;
51.     assign _and = R_type & f3_7 & ~funct7;
52.     assign _or  = R_type & f3_6 & ~funct7;
53.     assign _xor = R_type & f3_4 & ~funct7;
54.     assign _sll = R_type & f3_1 & ~funct7;
55.     assign _srl = R_type & f3_5 & ~funct7;
56.     assign _sra = R_type & f3_5 &  funct7;
57.     assign _slt = R_type & f3_2 & ~funct7;
58.     assign _sltu = R_type & f3_3 & ~funct7;
59.
60.     assign _addi = I_type & f3_0 & opcode[4];
61.     assign _andi = I_type & f3_7;
62.     assign _ori  = I_type & f3_6;
63.     assign _xori = I_type & f3_4 & opcode[4];
64.     assign _slti = I_type & f3_2 & opcode[4];
65.     assign _sltiu = I_type & f3_3;
66.     assign _slli = I_type & f3_1 & ~funct7 & opcode[4];
67.     assign _srli = I_type & f3_5 & ~funct7 & opcode[4];
68.     assign _srai = I_type & f3_5 & funct7 & opcode[4];
```

```
69.
70.     assign _lw   = I_type & f3_2 & ~opcode[4];
71.     assign _lb   = I_type & f3_0 & ~opcode[4];
72.     assign _lbu  = I_type & f3_4 & ~opcode[4];
73.     assign _lh   = I_type & f3_1 & ~opcode[4];
74.     assign _lhu  = I_type & f3_5 & ~opcode[4];
75.
76.     assign _sb = S_type & f3_0;
77.     assign _sh = S_type & f3_1;
78.     assign _sw = S_type & f3_2;
79.
80.     assign _lui   = U_type &  opcode[5];
81.     assign _auipc = U_type & ~opcode[5];
82.
83.     assign _jal  = J_type &  opcode[3];
84.     assign _jalr = J_type & ~opcode[3] & f3_0;
85.
86.     assign _beq = B_type & f3_0;
87.     assign _bne = B_type & f3_1;
88.     assign _blt = B_type & f3_4;
89.     assign _bge = B_type & f3_5;
90.     assign _bltu = B_type & f3_6;
91.     assign _bgeu = B_type & f3_7;
92.
93.     // R/I/U/J 类指令需要写寄存器堆
94.     assign reg_we = R_type | I_type | U_type | J_type;
95.
96.     // S 类指令需要写(RAM+外设)
97.     assign mem_we = S_type;
98.
99.     // 对应 mem_op 如下:
100.      // lb->0 lh->1 lw->2 lbu->3 lhu->4 sb->5 sh->6 sw->7
101.      assign mem_op[2] = _lhu | _sb  | _sh | _sw;
102.      assign mem_op[1] = _lw  | _lbu | _sh | _sw;
103.      assign mem_op[0] = _lh  | _lbu | _sb | _sw;
104.
105.      // 根据 ALU 操作码的设计决定，我的设计就是这样
106.      assign alu_op[3] = _sra | _srai | _bne | _slt | _slti |
107.                         _bge | _sltu | _sltiu | _bgeu | _blt |
108.                         _bltu | _lui | _auipc;
109.      assign alu_op[2] = _or | _ori | _xor | _xori | _beq | _sll |
110.                         _slli | _srl | _srli | _blt | _bltu |
```

```
                        _lui | _auipc;
    assign alu_op[1] = _sub | _and | _andi | _sll | _slli |
                        _srl | _srli | _slt | _slti | _bge |
                        _sltu | _sltiu | _bgeu | _lui | _auipc;
    assign alu_op[0] = _add | _addi | _lb | _lh | _lw | _lbu |
                        _lhu | _sb | _sh | _sw | _jalr | _and |
                        _andi | _xor | _xori | _beq | _srl |
                        _srli | _bne | _sltu | _sltiu | _bgeu |
                        _bltu | _auipc;

    assign use_imm = I_type | S_type | U_type | _jalr;

    assign use_imm20 = U_type;

    assign from_pc = _auipc;

    assign from_mem = _lb | _lh | _lw | _lbu | _lhu;

    assign from_pc4 = J_type;

    assign pc_next_select[1] = J_type;
    assign pc_next_select[0] = _jalr | (cond & B_type);

    initial begin
        $monitor($time,, "CU->cond => %b", cond);
        $monitor($time,, "CU->pc_next_select => %2b", pc_next_sele
ct);
    end
endmodule
```

IOMapping.v ： IO 管理模块

```
`timescale 1ns / 1ps
/*
 * 外设与 RAM 使用内存映射风格访问
 * addr=0x40 时从 switch 外设读数据
 * addr=0x80 时写数据至外设 result
 * 其余均为访问 RAM
 */
module IOMapping(
    // RAM
    input clk,
```

```
    input we,
    input [7:0] addr,
    input [31:0] wdata,
    input [2:0] mm,
    output [31:0] rdata,
    // IO
    input [5:0] switch,
    output reg [31:0] result
);
    wire [31:0] RAM_rdata, RAM_wdata;
    wire RAM_ce, RAM_we;
    assign RAM_ce = ~(|addr[7:6]);
    assign RAM_we = RAM_ce & we;
    RAM IOMapping_RAM(
        .clk(clk),
        .we(RAM_we),
        .mm(mm),
        .addr(addr[5:0]),
        .rdata(RAM_rdata),
        .raw_wdata(RAM_wdata)
    );

    assign rdata = addr[6] ? {26'h0, switch} : RAM_rdata;
    assign RAM_wdata = wdata;

    always @(posedge clk)
    begin
        if(we & addr[7])
            result <= wdata;
    end
endmodule
```

RAM.v ： 随机存取存储器模块

```
`timescale 1ns / 1ps

module RAM(
    input clk,
    input we,
    input [2:0] mm,
    input [5:0] addr,
    output reg [31:0] rdata,
```

```
9.      input [31:0] raw_wdata
10. );
11.     parameter LB = 0, LH = 1, LW = 2, LBU = 3, LHU = 4;
12.     // 用寄存器堆模拟 RAM
13.     reg [31:0] data[0:63];
14.
15.     // 未处理读出的数据 / 实际写的数据
16.     wire [31:0] raw_data, wdata;
17.
18.     assign raw_data = data[addr];
19.
20.     always @(*) begin
21.         case(mm)
22.             LB:  rdata = {{24{raw_data[7]}}, raw_data[7:0]};
23.             LH:  rdata = {{16{raw_data[15]}}, raw_data[15:0]};
24.             LW:  rdata = raw_data;
25.             LBU: rdata = {24'b0, raw_data[7:0]};
26.             LHU: rdata = {16'b0, raw_data[15:0]};
27.             default: rdata = 32'h0;
28.         endcase
29.     end
30.     // mm = 5 => sb
31.     // mm = 6 -> sh
32.     // mm = 7 => sw
33.     assign wdata = (
34.         {24'b0, {8{mm[2] & ~mm[1] & mm[0]}}}  |
35.         {16'b0, {16{(&mm[2:1]) & ~mm[0]}}} |
36.         {32{(&mm[2:0])}}
37.     ) & raw_wdata;
38.
39.     always @(posedge clk) begin
40.         if(we) begin
41.             data[addr] <= wdata;
42.         end
43.     end
44.
45.     initial begin
46.         $monitor($time, ,
47.             "RAM->we => %b\t->mm => %b\t->addr => 0x%8h\t->raw_wdata => 0x%8
    h",
48.             we,
49.             mm,
```

```
            addr,
            raw_wdata
        );
    end
endmodule
```

RegStack.v: 寄存器堆模块

```
`timescale 1ns / 1ps

/*
 * RISC-V 寄存器堆 数目：32 双端口读 上升沿写
 */
module RegStack(
    input clk,
    input we,               // 写使能 高电平有效
    input [4:0] raddr1,
    output [31:0] rdata1,
    input [4:0] raddr2,
    output [31:0] rdata2,
    input [4:0] waddr,
    input [31:0] wdata
);
    reg[31:0] regs[1:31];

    assign rdata1 = (raddr1 == 5'b00000) ? 32'b0 : regs[raddr1];
    assign rdata2 = (raddr2 == 5'b00000) ? 32'b0 : regs[raddr2];

    always @(posedge clk)
        if(we)
            if(waddr != 5'b00000)
                regs[waddr] <= wdata;

    initial begin
        $monitor($time, , "Reg->waddr => %d", waddr);
        $monitor($time, , "Reg->wdata => 0x%8h", wdata);
    end
endmodule
```

digit.v: 数码管模块

```
`timescale 1ns / 1ps
```

```
2.
3.  module digit(
4.      input [15:0] data,
5.      input clk,
6.      output reg[6:0] a2g,
7.      output reg[3:0] an
8.      );
9.      reg[1:0] status = 2'b00;
10.     reg[3:0] digit;
11.
12.     // 调度数码管
13.     always @(posedge clk)
14.         status<=status+1'b1;
15.
16.     // 根据不同状态来确定显示高 4 位还是低 4 位
17.     always @(*)
18.     case(status)
19.         2'b00:begin digit = data[15:12]; an=4'b1000;end
20.         2'b01:begin digit = data[11:8];  an=4'b0100;end
21.         2'b10:begin digit = data[7:4];   an=4'b0010;end
22.         2'b11:begin digit = data[3:0];   an=4'b0001;end
23.
24.         default:begin digit=data[3:0]; an=4'b0001;end
25.     endcase
26.
27.     //根据数字 digit 来设置不同的 a~g 段
28.     always @(*)
29.     case(digit)
30.             4'h0:a2g=7'b1111110;
31.             4'h1:a2g=7'b0110000;
32.             4'h2:a2g=7'b1101101;
33.             4'h3:a2g=7'b1111001;
34.             4'h4:a2g=7'b0110011;
35.             4'h5:a2g=7'b1011011;
36.             4'h6:a2g=7'b1011111;
37.             4'h7:a2g=7'b1110000;
38.             4'h8:a2g=7'b1111111;
39.             4'h9:a2g=7'b1111011;
40.             4'hA:a2g=7'b1110111;
41.             4'hB:a2g=7'b0011111;
42.             4'hC:a2g=7'b1001110;
43.             4'hD:a2g=7'b0111101;
```

```
            4'hE:a2g=7'b1001111;
            4'hF:a2g=7'b1000111;
            default:a2g=7'b1111110;
    endcase
endmodule
```

divclk.v: 分频器模块

```
`timescale 1ns / 1ps

module divclk(
    input clk,
    output new_clk
    );
    reg[18:0] data=19'b0;
    always @(posedge clk)
        data<=data+1'b1;
    // 100MHz 分频至 3Hz
    assign new_clk = data[18];
endmodule
```

四、 仿真程序

```
`timescale 1ns / 1ps

module sim(

);
    reg clk = 1'b0;
    always #5
        clk = ~clk;

    reg rst = 1'b1;
    initial #15
        rst = 1'b0;

    TOP sim_TOP(
        .clk(clk),
        .rst(rst),
        .n(6'hf)    // n 别超过 15 仿真时间不够运算 或者你可以调大仿真时间
    );
```

```
endmodule
```

五、 测试程序

R-type 测试程序

```
30'h0: ins = 32'hffe00093; // addi x1, x0, -2
30'h1: ins = 32'h00600113; // addi x2, x0, 6
30'h2: ins = 32'h002081b3; // add x3, x1, x2
30'h3: ins = 32'h40208233; // sub x4, x1, x2
30'h4: ins = 32'h0020f2b3; // and x5, x1, x2
30'h5: ins = 32'h0020e333; // or  x6, x1, x2
30'h6: ins = 32'h0020c3b3; // xor x7, x1, x2
30'h7: ins = 32'h00209433; // sll x8, x1, x2
30'h8: ins = 32'h0020d4b3; // srl x9, x1, x2
30'h9: ins = 32'h4020d533; // sra x10, x1, x2
30'ha: ins = 32'h0020a5b3; // slt x11, x1, x2
30'hb: ins = 32'h00112633; // slt x12, x2, x1
30'hc: ins = 32'h001136b3; // sltu x13, x2, x1
30'hd: ins = 32'h0020b733; // sltu x14, x1, x2
```

I-type 测试程序

```
30'h0: ins = 32'h123450b7; // lui x1, 0x12345
30'h1: ins = 32'h7890e093; // ori x1, x1, 0x789
30'h2: ins = 32'hffb00113; // addi x2, x0, -5
30'h3: ins = 32'h00104193; // xori x3, x0, 1
30'h4: ins = 32'h0090f213; // andi x4, x1, 9
30'h5: ins = 32'h00109293; // slli x5, x1, 1
30'h6: ins = 32'h0010d313; // srli x6, x1, 1
30'h7: ins = 32'h40115393; // srai x7, x2, 1
30'h8: ins = 32'h00312413; // slti x8, x2, 3
30'h9: ins = 32'h0090a493; // slti x9, x1, 9
30'ha: ins = 32'h00313b13; // sltiu x22, x2, 3
30'hb: ins = 32'h0051b593; // sltiu x11, x3, 5
30'hc: ins = 32'h00100503; // lb   x10, 1(x0)
30'hd: ins = 32'h00201583; // lh   x11, 2(x0)
30'he: ins = 32'h00402603; // lw   x12, 4(x0)
30'hf: ins = 32'h00804683; // lbu  x13, 8(x0)
30'h10: ins = 32'h01005703; // lhu  x14, 16(x0)
```

S-type 测试程序

```
30'h0: ins = 32'h124380b7; // lui x1, 0x12438
30'h1: ins = 32'h7ab0e093; // ori x1, x1, 0x7ab
30'h2: ins = 32'h00100023; // sb x1,0(x0)
30'h3: ins = 32'h001010a3; // sh x1,1(x0)
30'h4: ins = 32'h00102123; // sw x1,2(x0)
```

B-type 测试程序

```
30'h0: ins = 32'h00100093; // addi x1, x0, 1
30'h1: ins = 32'h7e100fe3; // beq x0, x1, 0xfff
30'h2: ins = 32'h00000463; // beq x0, x0, 8
30'h3: ins = 32'h00000033; // add x0, x0, x0
30'h4: ins = 32'h7e001fe3; // bne x0, x0, 0xfff
30'h5: ins = 32'h00101463; // bne x0, x1, 8
30'h6: ins = 32'h00000033; // add x0, x0, x0
30'h7: ins = 32'h7e00cfe3; // blt x1, x0, 0xfff
30'h8: ins = 32'h7e00efe3; // bltu x1, x0, 0xfff
30'h9: ins = 32'h7e105fe3; // bge x0, x1, 0xfff
30'ha: ins = 32'h7e107fe3; // bgeu x0, x1, 0xfff
```

U-type 测试程序

```
30'h0: ins = 32'h123450b7; // lui x1, 0x12345
30'h1: ins = 32'h10000117; // auipc x2, 0x10000
```

J-type 测试程序

```
30'h0: ins = 32'h0100006f; // jal x0, 0x10
30'h1: ins = 32'h001000b3; // add x1, x0, x1
30'h2: ins = 32'h001000b3; // add x1, x0, x1
30'h3: ins = 32'h001000b3; // add x1, x0, x1
30'h4: ins = 32'h01000067; // jalr x0, x0, 0x10
30'h5: ins = 32'h001000b3; // add x1, x0, x1
30'h6: ins = 32'h001000b3; // add x1, x0, x1
30'h7: ins = 32'h001000b3; // add x1, x0, x1
```

六、 仿真结果

R 类指令测试结果（按测试程序指令顺序）

```
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
   5 ROM->addr => 0x00000000
   5 PC->next => 0x00000004
   5 ROM->ins  => 0xffe00093
   5 Reg->waddr =>  1
   5 CU->pc_next_select => 00
   5 RAM->we => 0   ->mm => 000 ->addr => 0x0000003e    ->raw_wdata => 0xxxxxxxxx
   5 ALU->a => 0x00000000   ->b => 0xfffffffe   ->op => 0001
   5 ALU->f => 0xfffffffe
   5 CU->cond => 0
   5 Reg->wdata => 0xfffffffe
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
  15 ROM->addr => 0x00000004
  15 PC->next => 0x00000008
  15 ROM->ins  => 0x00600113
  15 Reg->waddr =>  2
  15 ALU->a => 0x00000000   ->b => 0x00000006   ->op => 0001
  15 ALU->f => 0x00000006
  15 RAM->we => 0   ->mm => 000 ->addr => 0x00000006    ->raw_wdata => 0xxxxxxxxx
  15 Reg->wdata => 0x00000006
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
  25 ROM->addr => 0x00000008
  25 PC->next => 0x0000000c
  25 ROM->ins  => 0x002081b3
  25 Reg->waddr =>  3
  25 ALU->a => 0xfffffffe   ->b => 0x00000006   ->op => 0001
  25 RAM->we => 0   ->mm => 000 ->addr => 0x00000004    ->raw_wdata => 0x00000006
  25 ALU->f => 0x00000004
  25 Reg->wdata => 0x00000004
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
    $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
      35 ROM->addr => 0x0000000c
      35 PC->next => 0x00000010
      35 ROM->ins  => 0x40208233
      35 Reg->waddr => 4
      35 ALU->a => 0xfffffffe   ->b => 0x00000006   ->op => 0010
      35 ALU->f => 0xfffffff8
      35 RAM->we => 0   ->mm => 000 ->addr => 0x00000038    ->raw_wdata => 0x00000006
      35 Reg->wdata => 0xfffffff8
    $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
      45 ROM->addr => 0x00000010
      45 PC->next => 0x00000014
      45 ROM->ins  => 0x0020f2b3
      45 Reg->waddr => 5
      45 ALU->a => 0xfffffffe   ->b => 0x00000006   ->op => 0011
      45 ALU->f => 0x00000006
      45 RAM->we => 0   ->mm => 000 ->addr => 0x00000006    ->raw_wdata => 0x00000006
      45 Reg->wdata => 0x00000006
    $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
      55 ROM->addr => 0x00000014
      55 PC->next => 0x00000018
      55 ROM->ins  => 0x0020e333
      55 Reg->waddr => 6
      55 ALU->a => 0xfffffffe   ->b => 0x00000006   ->op => 0100
      55 ALU->f => 0xfffffffe
      55 RAM->we => 0   ->mm => 000 ->addr => 0x0000003e    ->raw_wdata => 0x00000006
      55 Reg->wdata => 0xfffffffe
    $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

addi x1, x0, -2

x1=-2

addi x2, x0, 6

x2=6

add x3, x1, x2

x3=-2+6=4

sub x4, x1, x2

x4 = -2-6 = -8

and x5, x1, x2

x5 = 0xF..FE & 06 = 0x6

or x6, x1, x2

x6 = 0xF..FE | 0x6 = 0xF..FE

```
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
  65 ROM->addr => 0x00000018
  65 PC->next => 0x0000001c
  65 ROM->ins  => 0x0020c3b3
  65 Reg->waddr =>  7
  65 ALU->a => 0xfffffffe   ->b => 0x00000006   ->op => 0101
  65 ALU->f => 0xfffffff8
  65 RAM->we => 0   ->mm => 000 ->addr => 0x00000038    ->raw_wdata => 0x00000006
  65 Reg->wdata => 0xfffffff8
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
  75 ROM->addr => 0x0000001c
  75 PC->next => 0x00000020
  75 ROM->ins  => 0x00209433
  75 Reg->waddr =>  8
  75 ALU->a => 0xfffffffe   ->b => 0x00000006   ->op => 0110
  75 ALU->f => 0xffffff80
  75 RAM->we => 0   ->mm => 000 ->addr => 0x00000000    ->raw_wdata => 0x00000006
  75 Reg->wdata => 0xffffff80
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
  85 ROM->addr => 0x00000020
  85 PC->next => 0x00000024
  85 ROM->ins  => 0x0020d4b3
  85 Reg->waddr =>  9
  85 ALU->a => 0xfffffffe   ->b => 0x00000006   ->op => 0111
  85 ALU->f => 0x03ffffff
  85 RAM->we => 0   ->mm => 000 ->addr => 0x0000003f    ->raw_wdata => 0x00000006
  85 Reg->wdata => 0x03ffffff
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

xor x7, x1, x2

x1 = -2
x2 = 6

sll x8, x1, x2

srl x9, x1, x2

```
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
  95 ROM->addr => 0x00000024
  95 PC->next => 0x00000028
  95 ROM->ins  => 0x4020d533
  95 Reg->waddr => 10
  95 ALU->a => 0xfffffffe   ->b => 0x00000006   ->op => 1000
  95 ALU->f => 0xffffffff
  95 Reg->wdata => 0xffffffff
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
 105 ROM->addr => 0x00000028
 105 PC->next => 0x0000002c
 105 ROM->ins  => 0x0020a5b3
 105 Reg->waddr => 11
 105 ALU->a => 0xfffffffe   ->b => 0x00000006   ->op => 1010
 105 ALU->f => 0x00000001
 105 RAM->we => 0   ->mm => 000 ->addr => 0x00000001    ->raw_wdata => 0x00000006
 105 Reg->wdata => 0x00000001
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
 115 ROM->addr => 0x0000002c
 115 PC->next => 0x00000030
 115 ROM->ins  => 0x00112633
 115 Reg->waddr => 12
 115 ALU->a => 0x00000006   ->b => 0xfffffffe   ->op => 1010
 115 RAM->we => 0   ->mm => 000 ->addr => 0x00000000    ->raw_wdata => 0xfffffffe
 115 ALU->f => 0x00000000
 115 CU->cond => 1
 115 Reg->wdata => 0x00000000
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

x1 = -2
x2 = 6

sra x10, x1, x2

slt x11, x1, x2

slt x12, x2, x1

```
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
 125 ROM->addr => 0x00000030
 125 PC->next => 0x00000034
 125 ROM->ins  => 0x001136b3
 125 Reg->waddr => 13
 125 ALU->a => 0x00000006   ->b => 0xfffffffe   ->op => 1011
 125 ALU->f => 0x00000001
 125 CU->cond => 0
 125 RAM->we => 0   ->mm => 000 ->addr => 0x00000001    ->raw_wdata => 0xfffffffe
 125 Reg->wdata => 0x00000001
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
 135 ROM->addr => 0x00000034
 135 PC->next => 0x00000038
 135 ROM->ins  => 0x0020b733
 135 Reg->waddr => 14
 135 ALU->a => 0xfffffffe   ->b => 0x00000006   ->op => 1011
 135 RAM->we => 0   ->mm => 000 ->addr => 0x00000000    ->raw_wdata => 0x00000006
 135 ALU->f => 0x00000000
 135 CU->cond => 1
 135 Reg->wdata => 0x00000000
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

x1 = -2
x2 = 6

sltu x13, x2, x1

sltu x14, x1, x2

I 类指令测试结果（按测试程序指令顺序）

```
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
  5 ROM->addr => 0x00000000
  5 PC->next => 0x00000004
  5 ROM->ins  => 0x123450b7   lui x1, 0x12345
  5 Reg->waddr => 1
  5 CU->pc_next_select => 00
  5 ALU->a => 0xxxxxxxxx   ->b => 0x00012345   ->op => 1110
  5 RAM->we => 0   ->mm => 000 ->addr => 0x00000000   ->raw_wdata => 0xxxxxxxxx
  5 ALU->f => 0x12345000
  5 CU->cond => 0
  5 Reg->wdata => 0x12345000   Correct
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
 15 ROM->addr => 0x00000004
 15 PC->next => 0x00000008
 15 ROM->ins  => 0x7890e093   ori x1, x1, 0x789
 15 ALU->a => 0x12345000   ->b => 0x00000789   ->op => 0100
 15 ALU->f => 0x12345789   Reg->waddr不变，故不打印
 15 RAM->we => 0   ->mm => 000 ->addr => 0x00000009   ->raw_wdata => 0xxxxxxxxx
 15 Reg->wdata => 0x12345789   Correct
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
 25 ALU->a => 0x00000000   ->b => 0xfffffffb   ->op => 0001
 25 ROM->addr => 0x00000008
 25 PC->next => 0x0000000c
 25 ROM->ins  => 0xffb00113   addi x2, x0, -5
 25 Reg->waddr => 2
 25 ALU->f => 0xfffffffb
 25 RAM->we => 0   ->mm => 000 ->addr => 0x0000003b   ->raw_wdata => 0xxxxxxxxx
 25 Reg->wdata => 0xfffffffb   Correct
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
 35 ROM->addr => 0x0000000c
 35 PC->next => 0x00000010
 35 ROM->ins  => 0x00104193   xori x3, x0, 1
 35 Reg->waddr => 3
 35 ALU->a => 0x00000000   ->b => 0x00000001   ->op => 0101
 35 RAM->we => 0   ->mm => 000 ->addr => 0x00000001   ->raw_wdata => 0x12345789
 35 ALU->f => 0x00000001
 35 Reg->wdata => 0x00000001
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
 45 ROM->addr => 0x00000010
 45 PC->next => 0x00000014
 45 ROM->ins  => 0x0090f213   andi x4, x1, 9
 45 Reg->waddr => 4
 45 ALU->a => 0x12345789   ->b => 0x00000009   ->op => 0011
 45 RAM->we => 0   ->mm => 000 ->addr => 0x00000009   ->raw_wdata => 0xxxxxxxxx
 45 ALU->f => 0x00000009
 45 Reg->wdata => 0x00000009
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
 55 ROM->addr => 0x00000014
 55 PC->next => 0x00000018
 55 ROM->ins  => 0x00109293   slli x5, x1, 1
 55 Reg->waddr => 5
 55 ALU->a => 0x12345789   ->b => 0x00000001   ->op => 0110
 55 RAM->we => 0   ->mm => 000 ->addr => 0x00000012   ->raw_wdata => 0x12345789
 55 ALU->f => 0x2468af12
 55 Reg->wdata => 0x2468af12
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

```
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
  65 ROM->addr => 0x00000018
  65 PC->next => 0x0000001c
  65 ROM->ins  => 0x0010d313
  65 Reg->waddr =>  6
  65 ALU->a => 0x12345789   ->b => 0x00000001   ->op => 0111
  65 ALU->f => 0x091a2bc4
  65 RAM->we => 0   ->mm => 000 ->addr => 0x00000004    ->raw_wdata => 0x12345789
  65 Reg->wdata => 0x091a2bc4
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
  75 ROM->addr => 0x0000001c
  75 PC->next => 0x00000020
  75 ROM->ins  => 0x40115393
  75 Reg->waddr =>  7
  75 ALU->a => 0xfffffffb   ->b => 0x00000401   ->op => 1000
  75 ALU->f => 0xfffffffd
  75 RAM->we => 0   ->mm => 000 ->addr => 0x0000003d    ->raw_wdata => 0x12345789
  75 Reg->wdata => 0xfffffffd
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
  85 ROM->addr => 0x00000020
  85 PC->next => 0x00000024
  85 ROM->ins  => 0x00312413
  85 Reg->waddr =>  8
  85 ALU->a => 0xfffffffb   ->b => 0x00000003   ->op => 1010
  85 RAM->we => 0   ->mm => 000 ->addr => 0x00000001    ->raw_wdata => 0x00000001
  85 ALU->f => 0x00000001
  85 Reg->wdata => 0x00000001
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
  95 ROM->addr => 0x00000024
  95 PC->next => 0x00000028
  95 ROM->ins  => 0x0090a493
  95 Reg->waddr =>  9
  95 ALU->a => 0x12345789   ->b => 0x00000009   ->op => 1010
  95 RAM->we => 0   ->mm => 000 ->addr => 0x00000000    ->raw_wdata => 0xxxxxxxxx
  95 ALU->f => 0x00000000
  95 CU->cond => 1
  95 Reg->wdata => 0x00000000
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
 105 RAM->we => 0   ->mm => 000 ->addr => 0x00000000    ->raw_wdata => 0x00000001
 105 ROM->addr => 0x00000028
 105 PC->next => 0x0000002c
 105 ROM->ins  => 0x00313b13
 105 Reg->waddr => 22
 105 ALU->a => 0xfffffffb   ->b => 0x00000003   ->op => 1011
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
 115 ROM->addr => 0x0000002c
 115 PC->next => 0x00000030
 115 ROM->ins  => 0x0051b593
 115 Reg->waddr => 11
 115 ALU->a => 0x00000001   ->b => 0x00000005   ->op => 1011
 115 RAM->we => 0   ->mm => 000 ->addr => 0x00000001    ->raw_wdata => 0x2468af12
 115 ALU->f => 0x00000001
 115 CU->cond => 0
 115 Reg->wdata => 0x00000001
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

srli x6, x1, 1

srai x7, x2, 1

0x111...1010 -> 0x111...1101 Correct

slti x8, x2, 3

-5 < 3 =>set 1 Correct

slti x9, x1, 9

0x123..789 > 9 set 0

sltiu x22, x2, 3

0xff..fb > 3 set 0

sltiu x11, x3, 5

1<5 set 1

```
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
 125 ROM->addr => 0x00000030
 125 PC->next => 0x00000034
 125 ROM->ins  => 0x00100503
 125 Reg->waddr => 10
 125 ALU->a => 0x00000000   ->b => 0x00000001   ->op => 0001
 125 RAM->we => 0   ->mm => 000 ->addr => 0x00000001   ->raw_wdata => 0x12345789
 125 Reg->wdata => 0xxxxxxxxx
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
 135 ROM->addr => 0x00000034
 135 PC->next => 0x00000038
 135 ROM->ins  => 0x00201583
 135 Reg->waddr => 11
 135 ALU->a => 0x00000000   ->b => 0x00000002   ->op => 0001
 135 RAM->we => 0   ->mm => 001 ->addr => 0x00000002   ->raw_wdata => 0xfffffffb
 135 ALU->f => 0x00000002
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
 145 ROM->addr => 0x00000038
 145 PC->next => 0x0000003c
 145 ROM->ins  => 0x00402603
 145 Reg->waddr => 12
 145 ALU->a => 0x00000000   ->b => 0x00000004   ->op => 0001
 145 RAM->we => 0   ->mm => 010 ->addr => 0x00000004   ->raw_wdata => 0x00000009
 145 ALU->f => 0x00000004
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
 155 ROM->addr => 0x0000003c
 155 PC->next => 0x00000040
 155 ROM->ins  => 0x00804683
 155 Reg->waddr => 13
 155 ALU->a => 0x00000000   ->b => 0x00000008   ->op => 0001
 155 RAM->we => 0   ->mm => 011 ->addr => 0x00000008   ->raw_wdata => 0x00000001
 155 ALU->f => 0x00000008
 155 Reg->wdata => 0x000000xx
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
 165 ROM->addr => 0x00000040
 165 PC->next => 0x00000044
 165 ROM->ins  => 0x01005703
 165 Reg->waddr => 14
 165 ALU->a => 0x00000000   ->b => 0x00000010   ->op => 0001
 165 RAM->we => 0   ->mm => 100 ->addr => 0x00000010   ->raw_wdata => 0xxxxxxxxx
 165 ALU->f => 0x00000010
 165 Reg->wdata => 0x0000xxxx
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

lb x10, 1(x0)

RAM(1)当前没写值，所以为x

lh x11, 2(x0)

RAM（2）没写值，所以为x

lw x12, 4(x0)

RAM（4）没写值，所以为x

lbu x13, 8(x0)

RAM均未写值，所以为x

lhu x14, 16(x0)

S 类指令测试结果（按测试程序指令顺序）

```
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
   5 ROM->addr => 0x00000000
   5 PC->next => 0x00000004
   5 ROM->ins  => 0x124380b7
   5 Reg->waddr =>  1
   5 CU->pc_next_select => 00
   5 ALU->a => 0xxxxxxxxx   ->b => 0x00012438   ->op => 1110
   5 RAM->we => 0   ->mm => 000 ->addr => 0x00000000    ->raw_wdata => 0xxxxxxxxx
   5 ALU->f => 0x12438000
   5 CU->cond => 0
   5 Reg->wdata => 0x12438000
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
  15 ROM->addr => 0x00000004
  15 PC->next => 0x00000008
  15 ROM->ins  => 0x7ab0e093
  15 ALU->a => 0x12438000   ->b => 0x000007ab   ->op => 0100
  15 ALU->f => 0x124387ab
  15 RAM->we => 0   ->mm => 000 ->addr => 0x0000002b    ->raw_wdata => 0xxxxxxxxx
  15 Reg->wdata => 0x124387ab
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
  25 ALU->a => 0x00000000   ->b => 0x00000000   ->op => 0001
  25 ROM->addr => 0x00000008
  25 PC->next => 0x0000000c
  25 ROM->ins  => 0x00100023
  25 Reg->waddr =>  0
  25 RAM->we => 1   ->mm => 101 ->addr => 0x00000000    ->raw_wdata => 0x124387ab
  25 ALU->f => 0x00000000
  25 CU->cond => 1
  25 Reg->wdata => 0x00000000
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$


$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
  35 ROM->addr => 0x0000000c
  35 PC->next => 0x00000010
  35 ROM->ins  => 0x001010a3
  35 Reg->waddr =>  1
  35 ALU->a => 0x00000000   ->b => 0x00000001   ->op => 0001
  35 ALU->f => 0x00000001
  35 RAM->we => 1   ->mm => 110 ->addr => 0x00000001    ->raw_wdata => 0x124387ab
  35 CU->cond => 0
  35 Reg->wdata => 0x00000001
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
  45 ROM->addr => 0x00000010
  45 PC->next => 0x00000014
  45 ROM->ins  => 0x00102123
  45 Reg->waddr =>  2
  45 ALU->a => 0x00000000   ->b => 0x00000002   ->op => 0001
  45 ALU->f => 0x00000002
  45 RAM->we => 1   ->mm => 111 ->addr => 0x00000002    ->raw_wdata => 0x124387ab
  45 Reg->wdata => 0x00000002
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

lui x1, 0x12438

ori x1, x1, 0x7ab

前两条指令为了初始化x1

sb x1,0(x0)

实际上写入0xab,但没有监视

sh x1,1(x0)

实际上写入0x87ab

sw x1,2(x0)

实际上写入0x123487ab

B 类指令测试结果（按测试程序指令顺序）

```
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
   5 ROM->addr => 0x00000000
   5 PC->next => 0x00000004
   5 ROM->ins  => 0x00100093      addi x1, x0, 1
   5 Reg->waddr =>  1
   5 CU->pc_next_select => 00
   5 RAM->we => 0   ->mm => 000 ->addr => 0x00000001    ->raw_wdata => 0xxxxxxxxx
   5 ALU->a => 0x00000000   ->b => 0x00000001   ->op => 0001
   5 ALU->f => 0x00000001
   5 CU->cond => 0                Initial x1 = 1
   5 Reg->wdata => 0x00000001
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
  15 RAM->we => 0   ->mm => 000 ->addr => 0x00000001    ->raw_wdata => 0x00000001
  15 ROM->addr => 0x00000004
  15 PC->next => 0x00000008      0≠1 PC+4
  15 ROM->ins  => 0x7e100fe3     beq x0, x1, 0xfff
  15 Reg->waddr => 31
  15 ALU->a => 0x00000000   ->b => 0x00000001   ->op => 0101
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
  25 ROM->addr => 0x00000008
  25 PC->next => 0x00000010      0=0 PC+8
  25 ROM->ins  => 0x00000463     beq x0, x0, 8
  25 Reg->waddr =>  8
  25 ALU->a => 0x00000000   ->b => 0x00000000   ->op => 0101
  25 RAM->we => 0   ->mm => 000 ->addr => 0x00000000    ->raw_wdata => 0x00000000
  25 ALU->f => 0x00000000
  25 CU->cond => 1
  25 CU->pc_next_select => 01
  25 Reg->wdata => 0x00000000
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
  35 ROM->addr => 0x00000010     0=0 PC+4
  35 PC->next => 0x00000014
  35 ROM->ins  => 0x7e001fe3     bne x0, x0, 0xfff
  35 Reg->waddr => 31
  35 ALU->a => 0x00000000   ->b => 0x00000000   ->op => 1001
  35 ALU->f => 0x00000001
  35 CU->cond => 0
  35 CU->pc_next_select => 00
  35 RAM->we => 0   ->mm => 000 ->addr => 0x00000001    ->raw_wdata => 0x00000000
  35 Reg->wdata => 0x00000001
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
  45 ROM->addr => 0x00000014     0≠1 PC+8
  45 PC->next => 0x0000001c
  45 ROM->ins  => 0x00101463     bne x0, x1, 8
  45 Reg->waddr =>  8
  45 ALU->a => 0x00000000   ->b => 0x00000001   ->op => 1001
  45 RAM->we => 0   ->mm => 000 ->addr => 0x00000000    ->raw_wdata => 0x00000001
  45 ALU->f => 0x00000000
  45 CU->cond => 1
  45 CU->pc_next_select => 01
  45 Reg->wdata => 0x00000000
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
  55 ROM->addr => 0x0000001c     1>0 PC+4
  55 PC->next => 0x00000020
  55 ROM->ins  => 0x7e00cfe3     blt x1, x0, 0xfff
  55 Reg->waddr => 31
  55 ALU->a => 0x00000001   ->b => 0x00000000   ->op => 1100
  55 RAM->we => 0   ->mm => 000 ->addr => 0x00000001    ->raw_wdata => 0x00000000
  55 ALU->f => 0x00000001
  55 CU->cond => 0
  55 CU->pc_next_select => 00
  55 Reg->wdata => 0x00000001
```

```
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
 65 ROM->addr => 0x00000020
 65 PC->next => 0x00000024
 65 ROM->ins  => 0x7e00efe3
 65 ALU->a => 0x00000001   ->b => 0x00000000   ->op => 1101
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
 75 ROM->addr => 0x00000024
 75 PC->next => 0x00000028
 75 ROM->ins  => 0x7e105fe3
 75 ALU->a => 0x00000000   ->b => 0x00000001   ->op => 1010
 75 RAM->we => 0   ->mm => 000 ->addr => 0x00000001    ->raw_wdata => 0x00000001
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
 85 ROM->addr => 0x00000028
 85 PC->next => 0x0000002c
 85 ROM->ins  => 0x7e107fe3
 85 ALU->a => 0x00000000   ->b => 0x00000001   ->op => 1011
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

1>0 PC+4

bltu x1, x0, 0xfff

0<1 PC+4

bge x0, x1, 0xfff

0<1 PC+4

bgeu x0, x1, 0xfff

U 类指令测试结果（按测试程序指令顺序）

```
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
  5 ROM->addr => 0x00000000
  5 PC->next => 0x00000004
  5 ROM->ins  => 0x123450b7
  5 Reg->waddr =>  1
  5 CU->pc_next_select => 00
  5 ALU->a => 0xxxxxxxxx   ->b => 0x00012345   ->op => 1110
  5 RAM->we => 0   ->mm => 000 ->addr => 0x00000000    ->raw_wdata => 0xxxxxxxxx
  5 ALU->f => 0x12345000
  5 CU->cond => 0
  5 Reg->wdata => 0x12345000
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
 15 ROM->addr => 0x00000004
 15 PC->next => 0x00000008
 15 ROM->ins  => 0x10000117
 15 Reg->waddr =>  2
 15 ALU->a => 0x00000004   ->b => 0x00010000   ->op => 1111
 15 RAM->we => 0   ->mm => 000 ->addr => 0x00000004    ->raw_wdata => 0x00000000
 15 ALU->f => 0x10000004
 15 Reg->wdata => 0x10000004
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

Correct

PC value = 0x4

auipc x2, 0x10000

0x10000<<12 + 0x4 Correct

J 类指令测试结果（按测试程序指令顺序）

```
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
  5 ROM->addr => 0x00000000
  5 PC->next => 0x00000010
  5 ROM->ins  => 0x0100006f
  5 Reg->waddr =>  0
  5 CU->pc_next_select => 10
  5 ALU->a => 0x00000000   ->b => 0xxxxxxxxx   ->op => 0000
  5 RAM->we => 0   ->mm => 000 ->addr => 0x00000000    ->raw_wdata => 0xxxxxxxxx
  5 Reg->wdata => 0x00000004
  5 ALU->f => 0x00000000
  5 CU->cond => 1
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
 15 ROM->addr => 0x00000010
 15 ROM->ins  => 0x01000067
 15 Reg->wdata => 0x00000014
 15 ALU->a => 0x00000000   ->b => 0x00000010   ->op => 0001
 15 CU->pc_next_select => 11
 15 ALU->f => 0x00000010
 15 CU->cond => 0
 15 RAM->we => 0   ->mm => 000 ->addr => 0x00000010    ->raw_wdata => 0xxxxxxxxx
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

PC+0x10 = 0x10

jal x0, 0x10

jalr x0, x0, 0x10

PC = 0x10 Dead Loop

斐波那契程序仿真结果

```
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
 715 ROM->addr => 0x00000030
 715 PC->next => 0x00000034
 715 ROM->ins  => 0x09202023
 715 Reg->waddr =>  0
 715 ALU->a => 0x00000000   ->b => 0x00000080   ->op => 0001
 715 RAM->we => 0   ->mm => 111 ->addr => 0x00000000   ->raw_wdata => 0x00000262
 715 ALU->f => 0x00000080
 715 Reg->wdata => 0x00000080
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
 725 ROM->addr => 0x00000034
 725 Screen->data => 0x00000262
 725 PC->next => 0x00000038
```

output: sw s2, 0x80(x0) //输出至Screen

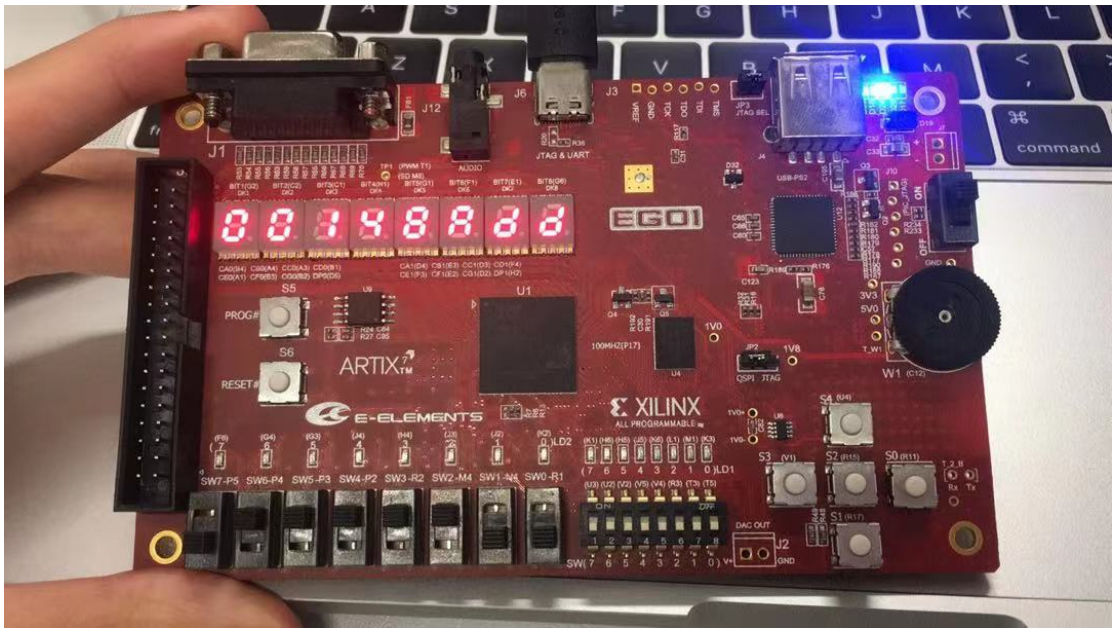
x0 + 0x80

IOMapping wdata 0x262

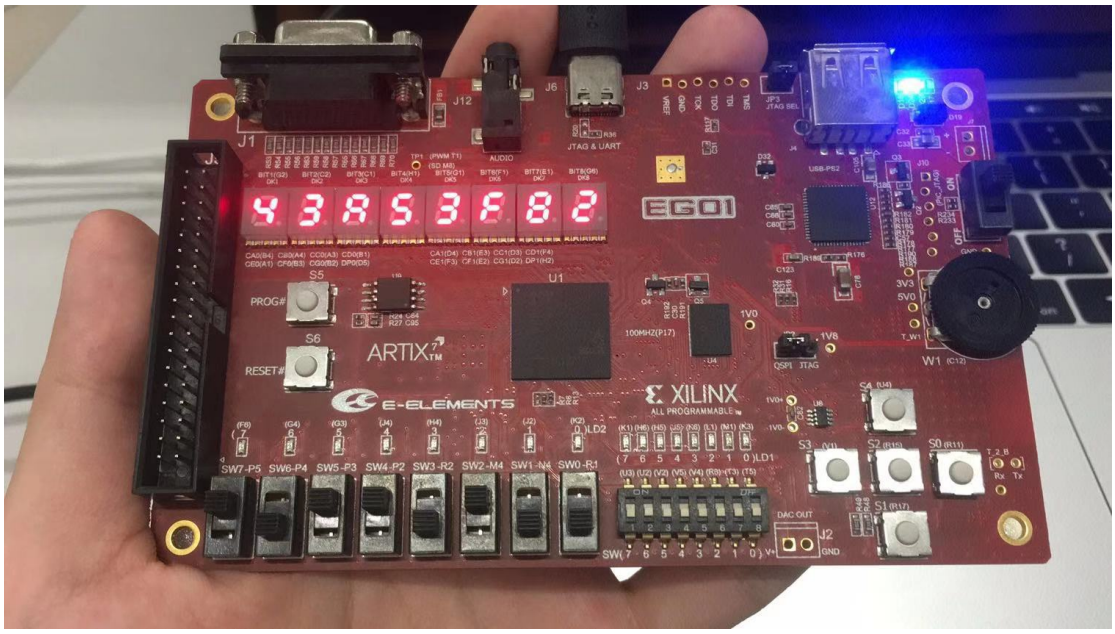
0x262 = 610 Correct

七、 实验结果

N = 0x1F = 31 时，数码管显示 0x148ADD。结果正确。



N = 0x2D = 45 时，数码管显示 0x43A53F82。结果正确。



八、 实验体会

通过本次实验，清晰的理解一条指令从 ROM 取出后进行执行的整个过程。总体来说，比前面几个实验增加了工作量，要去费心思看指令的功能。。

本次实验要注意的地方就在于要分析每条指令的功能，从而分析出每条指令在整个执行过程需要用到哪些部件，才能设计好控制器产生控制信号这层逻辑。

还有一个注意点就是分析每条指令的功能，从而设计出合适的 ALU 功能，为每一条指令设计特定的功能会造成功能冗余，比如 add 和 addi 指令都是只用到了 ALU 的 ADD 操作。因此对整个 RVI32 指令集分析后刚好得到 15 种不同的操作，刚好可以用 4 位比特表示。

其次呢，要理解好老师为什么要设计 IO 内存映射。由于我们之前的斐波那契程序是使用 ecall 指令来获取用户输入和数据输出的，本实验我们没有对 ecall 指令进行模拟。因此，为了实现用户输入，自然就是想通过开发版的 6 个开关进行模拟。那么我们如何将 6 个开关所产生的 6 位二进制数送到寄存器呢？注意到我们实现了 lw 指令，这指令的功能就是将某内存的字复制到某寄存器中，这是通过内存寻址实现的。我们将外设抽象成内存，也对他进行寻址，那么我们可以分配一个内存地址范围之外的地址给他。本实验采用 0x40 代表访问此外设，因此我们需要设计一个 RAM 的包装器，当地址大于内存寻址时就应该寻找外设的数据而不是内存。

最后就是注意左右四个数码管是分开显示的，这是由于开发版的原因。测试也没什么心得哈哈，我就调了一次（控制器的指令信号错误使能）就成功了。