实验报告

2021 年 6 月 9 日　　成绩：_____

姓名	刘合	学号		班级	
专业	计算机科学与技术		课程名称	计算机组成原理课程设计	
任课老师	冯建文	指导老师		机位号	
实验序号	6	实验名称	取指令及指令译码实验		
实验时间	2021.6.9	实验地点		实验设备号	

一、实验目的与要求

1、 实验目的：

1.掌握指令存储器、 PC 与 IR 的设计方法;

2.掌握 CPU 取指令操作与指令译码的方法和过程，掌握指令译码器的设计方法;

3.理解 RISC-V 立即数的生成与扩展方法，掌握立即数拼接与扩展器的设计。

2、 实验要求：

1. 用实验 5 的 acc.s 或 move.s 的机器指令代码，构造一个 acc.coe 或 move.coe 文件。

2.使用 Vivado 或者 ISE 的 MemoryIP 核,新建一个 64X32 位只读存储器作为指令存储器 IM，并关联 acc.coe 或 move.coe 文件，初始化其内容。

3. 构造 PC 和 IR 寄存器,并与上述指令存储器 IM 连接,构成取指令模块,如图 11.52 所示

4.对取指令 模块进行仿真，确保能正确取出指令，并且 PC 自增。

5. 编写指令译码模块 IDI，包括立即数拼接与扩展器模块 ImmU。

6.对 ImmU 模块进行仿真，确保立即数拼接与扩展正确。

7.连接取指令模块和指令译码模块 IDI,如图所示。

8.针对使用的实验板卡， 设计取指令与指令译码模块的板级验证实验方案,编写项层测试模块。表 11.12 给出了本实验 FPGA 引脚信号配置的一个建议，需要 32 位显示灯，不一.定适合所有的板卡。

表11.12 实验 6 信号配置表

	信号名称	配置设备管脚

输入	PC_Write	逻辑开关
	IR_Write	逻辑开关
时钟	rst_n	按键
	clk_im	按键
输出	imm32[31:0]	8 个数码管
	rs1[4:0]	5 个 LED 灯
	rs2[4:0]	5 个 LED 灯
	rd[4:0]	5 个 LED 灯
	opcode[6:0]	7 个 LED 灯
	funct3[2:0]	3 个 LED 灯
	funct7[6:0]	7 个 LED 灯

图 11.56 给出了另一种板级验证方案，为便于观察，将 PC 和 IR 的内容从顶层模块的端口输出，和生成的立即数一起送 到数码管显示，由两位开关进行三选一选择输出；将解析出的各字段,送到显示灯输出。请参考实验 2 中的板级验证设计方法，根据板卡资源自行修改设计。

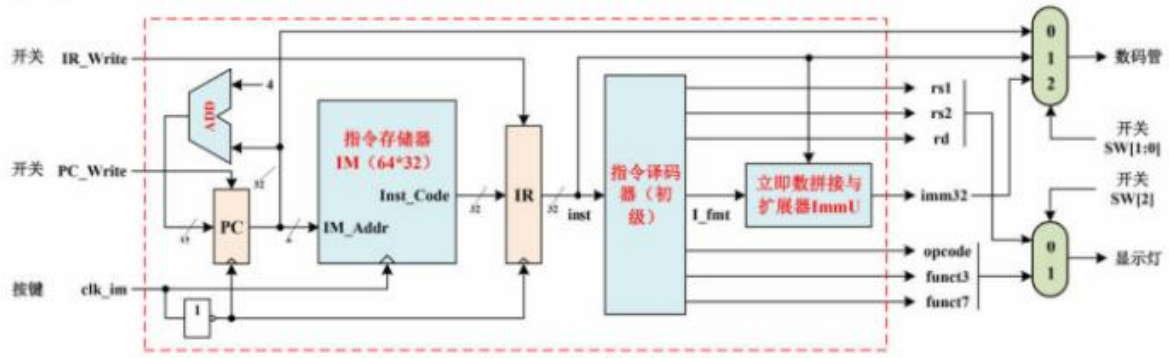


图11.56 实验 6 的顶层测试模块范例

9. 复位后，按 clk_im 按键，执行取指令操作，验证读出的指令代码是否和 COE 文件一致,生成的立即数是否正确，解析的各字段是否正确，将实验结果记录到表中。

10.撰写实验报告，格式见附录，重点内容包括:对仿真结果进行分析;描述你设计的板级验证实验方案、模块结构与连接;说明你的板级操作过程;分析记录下来的板级实验结果、得到有效结论。请力所能及回答或实践本实验的“思考与探索”部分。

3.实验步骤

1. 新建一个 acc.coe 或 move.coe 文件，然后打开实验 5 的 acc.s 或 move.s 汇编语言程序的反汇编文件，将其中的 32 位机器指令代码逐个拷贝到.coe 文件中，作为指令存储器的初始化数据。

2.新建一个工程， 通过 IP 核生成向导创建一一个 64X32 位的单端口 ROM 存储器，步骤同实验 4,但是 Memory Type 选择 Single Port ROM 类型即可;初始化文件选择前述编辑的 acc .coe 或 move.coe。

3.编写 PC 寄存器模块和 IR 寄存器模块:复位时清零;打入脉冲来临时，如果写信号有效则执行打入操作。

4. 生成 ROM 存储器的实例作为指令存储器 IM 模块，并与上述 PC 和 IR 模块连接，构成取指令模块 IM_ IF， 如图 11.52 所示。

5. 编写测试代码，对取指令模块进行仿真，清零后，从 0 号单元开始顺序取指令:当 PC_ Write 和 IR_ Write 保持为 1 时，每来一个 clk_im，就从 inst 输出一条指令，并且 PC+4.

6. 按照图 11.54 以及范例，编写立即数拼接与扩展器模块 ImmU， 它是组合逻辑电路。

7. 编写测试代码，对 ImmU 模块进行仿真，确保立即数拼接与扩展正确。

8. 编写指令译码模块 IDI (如图 11.53),对指令格式进行译码，并调用立即数拼接与扩展器模块 ImmU。

9.如图 11.55 所示，构造取指令与译码模块 IM_ IF_ ID，调用取指令模块和指令译码模块，并连接它们。

10.依据表 11.12 和图 11.56 的提示,以及实际板卡情况,设计取指令与译码模块 IM_ _IF_ ID 的板级验证实验方案，然后据此编写一个顶层测试模块。

11. 新建管脚约束文件，进行相应的引脚配置。

12. 生成*.bit 文件，下载到实验设备的 FPGA 芯片中。

13. 板级实验:按照你所设计的实验方案，操作输入设备、观察输出设备，预期的验证操作如下:

(1)按 rst_n 按键，将 PC 和 IR 清零，接下来将从 0 号单元开始取指令;

(2)拨开关 PC_ _Write=1. IR_ Write=1， 按动时钟键 clk_im，每按一下， 就取出一条指令;

(3)观察输出设备上读出的指令是否按序读出,代码是否与初始化的 coe 文件中的指令码一致;

二、实验设计与程序代码

1、 模块设计说明

程序包含了 7 个模块一个存储器 IP 核：

1.顶层模块：有序调用其他模块的程序；

2.取指令模块：从指令存储器中依次取出指令，传输给译码模块

3.译码模块：把取出的指令解析输出

4.显示灯模块：显示指令译码输出的各个部分数据

5.分频器模块：是为了给数码管扫描显示模块分配时钟频率；

6.数码管扫描显示模块：是为了把 BCD 码数据以十进制数字显示出来

7.数据选择模块：选择不同的值存储到存储器。

8. 存储器 IP 核：存储指令数据

2.实验程序源代码及注释等

```
module dingceng(CLR,clk_1M,SW,K,rst_n,IR_Write,PC_Write,clk_im,LED,AN,seg);
    input CLR;
    input clk_1M;
    input [1:0]SW;
    input K;
    input rst_n;
    input IR_Write;
    input PC_Write;
    input clk_im;
    output [16:0]LED;
    output [7:0]AN,seg;
    wire [4:0]rs1;
    wire [4:0]rs2;
    wire [4:0]rd;
    wire [6:0]opcode;
    wire [2:0]funct3;
    wire [6:0]funct7;
    wire [31:0]PC,inst,imm32,F;
    wire CLK;
```

```
    zhilin oo(rst_n,IR_Write,PC_Write,clk_im,inst,PC);
    yima pp(inst,rs1,rs2,rd,imm32,opcode,funct3,funct7);
    threeone qq(SW,PC,inst,imm32,F);
    xsd rr(K,rs1,rs2,rd,opcode,funct3,funct7,LED[16:0]);
    fpq ss(CLR,clk_1M,CLK);
    smgsm tt(CLK,CLR,F,AN,seg);
endmodule

module zhilin(rst_n,IR_Write,PC_Write,clk_im,inst,PC);
    input rst_n;
    input IR_Write;
    input PC_Write;
    input clk_im;
    output [31:0]inst;
    output [31:0]PC;
    wire clk_fim;
    wire [31:0]PC_Out;
    wire [31:0]Inst_Code;
    assign clk_fim=~clk_im;
    PC aa(rst_n,PC_Write,clk_fim,PC_Out);
    assign PC=PC_Out;
    RAM_B IM(         //存储器
   .clka(clk_im),       // input wire clka
   .addra(PC_Out[7:2]),    // input wire [5 : 0] addra
   .douta(Inst_Code)    // output wire [31 : 0] douta
   );
    IR cc(Inst_Code,clk_fim,IR_Write,inst);
Endmodule
```

```
module PC(rst_n,PC_Write,clk_fim,PC_Out);//程序计数器 PC
    input rst_n;
    input PC_Write;
    input clk_fim;
    output reg [31:0]PC_Out;
    initial
        begin
            PC_Out<=0;
        end
    always@(posedge clk_fim or posedge rst_n)
    begin
        if(rst_n==0)
            PC_Out<=0;
        else
            begin
                if(PC_Write==1)
                    PC_Out<=PC_Out+4;
                else
                    PC_Out<=PC_Out;
            end
    end
endmodule

module IR(Inst_Code,clk_fim,IR_Write,inst);
    input [31:0]Inst_Code;
    input clk_fim;
    input IR_Write;
    output reg [31:0]inst;
    always@(posedge clk_fim)
    begin
```

```
        if(IR_Write==1)
            inst<=Inst_Code;
        end
endmodule

module yima(inst,rs1,rs2,rd,imm32,opcode,funct3,funct7);
         input [31:0]inst;
         output [4:0]rs1;
         output [4:0]rs2;
         output [4:0]rd;
         output [31:0]imm32;
         output [6:0]opcode;
         output [2:0]funct3;
         output [6:0]funct7;
         wire [6:0]I_fmt;
         ID1 dd(inst,rs1,rs2,rd,I_fmt,opcode,funct3,funct7);
         ImmU ee(inst,I_fmt,imm32);
Endmodule

module ID1(inst,rs1,rs2,rd,I_fmt,opcode,funct3,funct7);
         input [31:0]inst;
         output [4:0]rs1;
         output [4:0]rs2;
         output [4:0]rd;
         output [6:0]I_fmt;
         output [6:0]opcode;
         output [2:0]funct3;
         output [6:0]funct7;
         wire R_Type,I_Type,S_Type,B_Type,U_Type,J_Type,YW_Type;
    assign I_fmt={R_Type,I_Type,S_Type,B_Type,U_Type,J_Type,YW_Type};
```

```
    `define OP_R    7'b0110011
    assign R_Type=(opcode==`OP_R);
    `define OP_I1    7'b0010011
    `define OP_I2    7'b0000011
    assign I_Type=(opcode==`OP_I1||opcode==`OP_I2);
    `define OP_S    7'b0100011
    assign S_Type=(opcode==`OP_S);
    `define OP_B    7'b1100011
    assign B_Type=(opcode==`OP_B);
    `define OP_U1    7'b0110111
    `define OP_U2    7'b0010111
    assign U_Type=(opcode==`OP_U1||opcode==`OP_U2);
    `define OP_J    7'b1101111
    assign J_Type=(opcode==`OP_J);
    `define OP_I3    7'b1100111
    assign YW_Type=(opcode==`OP_I3);
    assign opcode=inst[6:0];
    assign rs1=inst[19:15];
    assign rs2=inst[24:20];
    assign rd=inst[11:7];
    assign funct3=inst[14:12];
    assign funct7=inst[31:25];
endmodule

module ImmU(inst,I_fmt,imm32);
        input [31:0]inst;
        input [6:0]I_fmt;
        output reg [31:0]imm32;
        wire [31:0]I_imm;
        wire [31:0]S_imm;
```

```
    wire [31:0]B_imm;
    wire [31:0]U_imm;
    wire [31:0]J_imm;
    wire [31:0]yw_imm;
    assign I_imm={{20{inst[31]}},inst[31:20]};
    assign S_imm={{20{inst[31]}},inst[31:25],inst[11:7]};
    assign B_imm={{20{inst[31]}},inst[7],inst[30:25],inst[11:8],1'b0};
    assign U_imm={inst[31:12],{12{1'b0}}};
    assign J_imm={{12{inst[31]}},inst[19:12],inst[20],inst[30:21],1'b0};
    assign yw_imm={{20{1'b0}},inst[11:0]};
    always@(*)
        begin
            case(I_fmt)
                7'b0100000: imm32=I_imm;//I
                7'b0010000: imm32=S_imm;//S
                7'b0001000: imm32=B_imm;//B
                7'b0000100: imm32=U_imm;//U
                7'b0000010: imm32=J_imm;//J
                7'b0000001:imm32=yw_imm;//I 型移位
                default:imm32=0;
            endcase
        end
endmodule

module threeone(SW,PC,inst,imm32,F);//三选一 A,B,F 输出到数码管显示
    input [1:0]SW;
    input [31:0]PC;
    input [31:0]inst;
    input [31:0]imm32;
    output reg [31:0]F;
```

```
    always@(*)
    begin
        case(SW)
            2'b00:begin F=PC; end
            2'b01:begin F=inst; end
            2'b10:begin F=imm32; end
            default:begin F=PC; end
        endcase
    end
endmodule

module xsd(K,rs1,rs2,rd,opcode,funct3,funct7,LED);
    input K;
    input [4:0]rs1,rs2,rd;
    input [6:0]opcode;
    input [2:0]funct3;
    input [6:0]funct7;
    output reg [16:0]LED;
    always@(*)
    begin
        case(K)
            0: begin
                LED[16:12]=rs1;
                LED[11:7]=rs2;
                LED[6:2]=rd;
                LED[1:0]=2'b00;
              end
            1: begin
                LED[16:10]=opcode;
                LED[9:7]=funct3;
```

```
                    LED[6:0]=funct7;
                end
        endcase
    end
endmodule

module fpq(CLR,clk_1M,CLK);              //分频器模块
   input CLR;
   input clk_1M;
   output reg CLK;
   reg [8:0]counter;
   always@(negedge CLR or posedge clk_1M)
      begin
         if(!CLR)
            begin
               counter<=9'd0;
               CLK<=1'b0;
            end
         else if(counter==9'd500)
               begin
                  CLK<=~CLK;
                  counter<=9'd0;
               end
         else
               counter<=counter+1'b1;
      end
endmodule

module smgsm(CLK,CLR,Choice,AN,seg);//数码管扫描显示模块
```

```
input CLK; //扫描的间隔始终，控制数码管轮流点亮的频率
input CLR; //复位信号，用于初始化状态
input[31:0]Choice;
output reg [7:0]AN;//位选
output reg [7:0]seg; //段选
reg [3:0] data;
reg [2:0] bit_sel;  //数码管计数器指示，000~111  最左到最右
always@(negedge CLR or posedge CLK)
begin
    if(!CLR)
            bit_sel<=3'b000;
    else
         bit_sel<=bit_sel+1'b1;
end
always@(*)
    begin
        case(bit_sel)
            3'b000:data<=Choice[3:0];
            3'b001:data<=Choice[7:4];
            3'b010:data<=Choice[11:8];
            3'b011:data<=Choice[15:12];
            3'b100:data<=Choice[19:16];
            3'b101:data<=Choice[23:20];
            3'b110:data<=Choice[27:24];
            3'b111:data<=Choice[31:28];
            default:data<=data;
        endcase
    end
always@(*)
    begin
```

```
        case(bit_sel)
            3'b000:AN=8'b11111110;
            3'b001:AN=8'b11111101;
            3'b010:AN=8'b11111011;
            3'b011:AN=8'b11110111;
            3'b100:AN=8'b11101111;
            3'b101:AN=8'b11011111;
            3'b110:AN=8'b10111111;
            3'b111:AN=8'b01111111;
            default:AN=8'b11111111;
        endcase
    end
always@(*)
    begin
        case(data)
                0:seg<=8'b00000011;
                  1:seg<=8'b10011111;
                  2:seg<=8'b00100101;
                  3:seg<=8'b00001101;
                  4:seg<=8'b10011001;
                  5:seg<=8'b01001001;
                  6:seg<=8'b01000001;
                  7:seg<=8'b00011111;
                  8:seg<=8'b00000001;
                  9:seg<=8'b00001001;
                  10:seg<=8'b00010001;
                  11:seg<=8'b11000001;
                  12:seg<=8'b01100011;
                  13:seg<=8'b10000101;
                  14:seg<=8'b01100001;
```

```
                15:seg<=8'b01110001;
            default:seg<=8'b11111111;
        endcase
    end
endmodule
```

三、实验仿真

1、 仿真代码

```
//下面是取指令模块的仿真激励代码
module qzfz();
//INPUT
reg rst_n;
reg IR_Write;
reg PC_Write;
reg clk_im;
//OUTPUT
wire [31:0]inst;
initial
    begin
        rst_n=1;clk_im=0;IR_Write=0;PC_Write=1;
    end
always
    begin
        #200    rst_n=~rst_n;
    end
always
    begin
        #25    clk_im=~clk_im;
    end
always
```

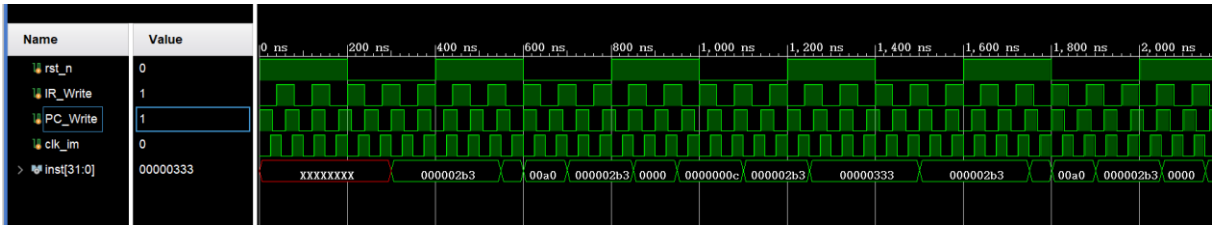
```
        begin
            #40    IR_Write=~IR_Write;
        end
always
        begin
            #30    PC_Write=~PC_Write;
        end
    zhilin ff(rst_n,IR_Write,PC_Write,clk_im,inst);
endmodule
//下面是译码模块的仿真激励代码
module ymfz();
//INPUT
reg [31:0]inst;
//OUTPUT
wire [4:0]rs1;
wire [4:0]rs2;
wire [4:0]rd;
wire [31:0]imm32;
wire [6:0]opcode;
wire [2:0]funct3;
wire [6:0]funct7;
initial
      begin
          inst=32'h0000_02b3;//add    R 型
          #50 inst=32'hfff3_8393;//addi I 型
          #50 inst=32'h09c0_2023;//SW      S 型
      end
    yima kk(inst,rs1,rs2,rd,imm32,opcode,funct3,funct7);
endmodule
RAM_B vv(
```

```
    .clka(clka),        // input wire clka
    .wea(wea),          // input wire [0 : 0] wea
    .addra(addra),    // input wire [5 : 0] addra
    .dina(dina),        // input wire [31 : 0] dina
    .douta(douta)    // output wire [31 : 0] douta
);
endmodule
```

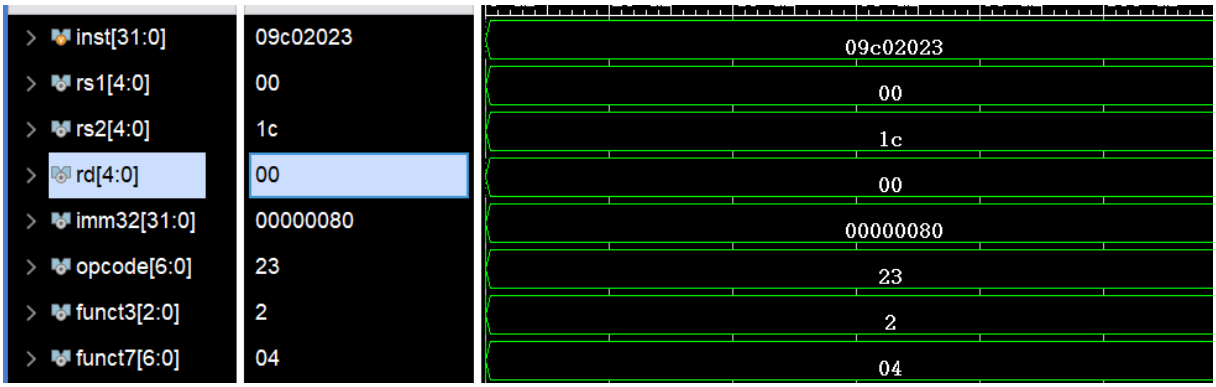
2、 仿真波形

（运行仿真时，波形截图）

取指令：



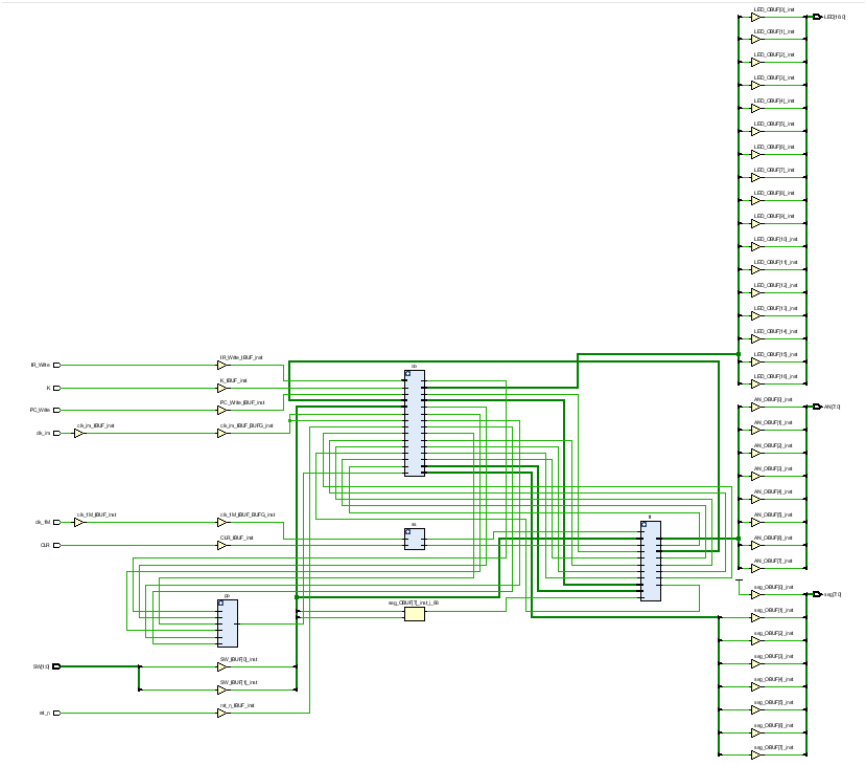
译码：



3、 仿真结果分析

取指令与译码功能都符合预期的效果。逻辑成功。

四、电路图



五、引脚配置

（引脚约束文件的内容，描述主要配置情况）

```
set_property IOSTANDARD LVCMOS18 [get_ports clk_im]
set_property IOSTANDARD LVCMOS18 [get_ports {AN[7]}]
set_property IOSTANDARD LVCMOS18 [get_ports {AN[6]}]
set_property IOSTANDARD LVCMOS18 [get_ports {AN[5]}]
set_property IOSTANDARD LVCMOS18 [get_ports {AN[4]}]
set_property IOSTANDARD LVCMOS18 [get_ports {AN[3]}]
set_property IOSTANDARD LVCMOS18 [get_ports {AN[2]}]
set_property IOSTANDARD LVCMOS18 [get_ports {AN[1]}]
set_property IOSTANDARD LVCMOS18 [get_ports {AN[0]}]
set_property IOSTANDARD LVCMOS18 [get_ports {LED[16]}]
set_property IOSTANDARD LVCMOS18 [get_ports {LED[15]}]
set_property IOSTANDARD LVCMOS18 [get_ports {LED[14]}]
set_property IOSTANDARD LVCMOS18 [get_ports {LED[13]}]
set_property IOSTANDARD LVCMOS18 [get_ports {LED[12]}]
set_property IOSTANDARD LVCMOS18 [get_ports {LED[11]}]
```

```
set_property IOSTANDARD LVCMOS18 [get_ports {LED[10]}]
set_property IOSTANDARD LVCMOS18 [get_ports {LED[9]}]
set_property IOSTANDARD LVCMOS18 [get_ports {LED[8]}]
set_property IOSTANDARD LVCMOS18 [get_ports {LED[7]}]
set_property IOSTANDARD LVCMOS18 [get_ports {LED[6]}]
set_property IOSTANDARD LVCMOS18 [get_ports {LED[5]}]
set_property IOSTANDARD LVCMOS18 [get_ports {LED[4]}]
set_property IOSTANDARD LVCMOS18 [get_ports {LED[3]}]
set_property IOSTANDARD LVCMOS18 [get_ports {LED[2]}]
set_property IOSTANDARD LVCMOS18 [get_ports {LED[1]}]
set_property IOSTANDARD LVCMOS18 [get_ports {LED[0]}]
set_property IOSTANDARD LVCMOS18 [get_ports {seg[7]}]
set_property IOSTANDARD LVCMOS18 [get_ports {seg[6]}]
set_property IOSTANDARD LVCMOS18 [get_ports {seg[5]}]
set_property IOSTANDARD LVCMOS18 [get_ports {seg[4]}]
set_property IOSTANDARD LVCMOS18 [get_ports {seg[3]}]
set_property IOSTANDARD LVCMOS18 [get_ports {seg[2]}]
set_property IOSTANDARD LVCMOS18 [get_ports {seg[1]}]
set_property IOSTANDARD LVCMOS18 [get_ports {seg[0]}]
set_property IOSTANDARD LVCMOS18 [get_ports {SW[1]}]
set_property IOSTANDARD LVCMOS18 [get_ports {SW[0]}]
set_property IOSTANDARD LVCMOS18 [get_ports clk_1M]
set_property IOSTANDARD LVCMOS18 [get_ports CLR]
set_property IOSTANDARD LVCMOS18 [get_ports IR_Write]
set_property IOSTANDARD LVCMOS18 [get_ports K]
set_property IOSTANDARD LVCMOS18 [get_ports PC_Write]
set_property IOSTANDARD LVCMOS18 [get_ports rst_n]
set_property PACKAGE_PIN E3 [get_ports clk_1M]
set_property PACKAGE_PIN N17 [get_ports CLR]
set_property PACKAGE_PIN U6 [get_ports {LED[16]}]
```

```
set_property PACKAGE_PIN R5 [get_ports {LED[15]}]
set_property PACKAGE_PIN U7 [get_ports {LED[14]}]
set_property PACKAGE_PIN R6 [get_ports {LED[13]}]
set_property PACKAGE_PIN R7 [get_ports {LED[12]}]
set_property PACKAGE_PIN U8 [get_ports {LED[11]}]
set_property PACKAGE_PIN T8 [get_ports {LED[10]}]
set_property PACKAGE_PIN V9 [get_ports {LED[9]}]
set_property PACKAGE_PIN T10 [get_ports {LED[8]}]
set_property PACKAGE_PIN V11 [get_ports {LED[7]}]
set_property PACKAGE_PIN T11 [get_ports {LED[6]}]
set_property PACKAGE_PIN V12 [get_ports {LED[5]}]
set_property PACKAGE_PIN R12 [get_ports {LED[4]}]
set_property PACKAGE_PIN U13 [get_ports {LED[3]}]
set_property PACKAGE_PIN R13 [get_ports {LED[2]}]
set_property PACKAGE_PIN U14 [get_ports {LED[1]}]
set_property PACKAGE_PIN T15 [get_ports {LED[0]}]
set_property PACKAGE_PIN C9 [get_ports {AN[7]}]
set_property PACKAGE_PIN C10 [get_ports {AN[6]}]
set_property PACKAGE_PIN D10 [get_ports {AN[5]}]
set_property PACKAGE_PIN C11 [get_ports {AN[4]}]
set_property PACKAGE_PIN M17 [get_ports {AN[3]}]
set_property PACKAGE_PIN J14 [get_ports {AN[2]}]
set_property PACKAGE_PIN K13 [get_ports {AN[1]}]
set_property PACKAGE_PIN P14 [get_ports {AN[0]}]
set_property PACKAGE_PIN F14 [get_ports {seg[7]}]
set_property PACKAGE_PIN N14 [get_ports {seg[6]}]
set_property PACKAGE_PIN J13 [get_ports {seg[5]}]
set_property PACKAGE_PIN G13 [get_ports {seg[4]}]
set_property PACKAGE_PIN F13 [get_ports {seg[3]}]
set_property PACKAGE_PIN G14 [get_ports {seg[2]}]
```

```
set_property PACKAGE_PIN M13 [get_ports {seg[1]}]
set_property PACKAGE_PIN H14 [get_ports {seg[0]}]
set_property PACKAGE_PIN U18 [get_ports rst_n]
set_property PACKAGE_PIN U17 [get_ports clk_im]
set_property PACKAGE_PIN V5 [get_ports {SW[1]}]
set_property PACKAGE_PIN T4 [get_ports {SW[0]}]
set_property PACKAGE_PIN V6 [get_ports K]
set_property PACKAGE_PIN R15 [get_ports PC_Write]
set_property PACKAGE_PIN U16 [get_ports IR_Write]
set_property PULLDOWN true [get_ports clk_1M]
set_property PULLDOWN true [get_ports CLR]
set_property PULLDOWN true [get_ports IR_Write]
set_property PULLDOWN true [get_ports K]
set_property PULLDOWN true [get_ports PC_Write]
set_property PULLDOWN true [get_ports rst_n]
set_property PULLDOWN true [get_ports {SW[1]}]
set_property PULLDOWN true [get_ports {SW[0]}]
set_property PULLDOWN true [get_ports clk_im]
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets clk_1M]
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets clk_im]
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets CLR]
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets rst_n]
```

六、思考与探索

1、 实验结果记录：

（实验操作的过程及结果记录）

PC	IR	COE 文件中指令代码	汇编指令	立即数 imm32	解析字段是否正确
0000_000c	00a0_0393	00a00393	li t2,10	0000_000a	正确
0000_0014	0403_2e03	04032e03	lw t3,64(t1)	0000_0040	正确
0000_0030	09c0_2023	09c02023	sw t3,128(zero) #80<Loop2+0x5c>	0000_0080	正确
0000_0024	0003_8463	00038463	beq zt2,24<Loop2>	0000_0008	正确
0000_0018	01c2_82b3	01c282b3	add t0,t0,t3	0000_0000	正确
0000_001c	0043_0313	00430313	addit1,t1,4	0000_0004	正确
0000_0028	fedf_f06f	fedff06f	j c <Loop1>	ffff_ffec	正确

2、 实验结论：

实验结果符合预期结果，逻辑正确，实验成功。

3、 问题与解决方案

编写译码块时测试时出现一直只出现 0 不变的情况，无法分析出正确的立即数。考虑不周全，忘了编写移位指令的逻辑，编写后，可以输出正常结果。

4、 思考题：

1. 在复位后，第一次按动 clk_im 按钮,你的程序读出的指令是哪个单元的? 0 号单元还是 4 号单元的指令?分析为什么?如果要求在复位后的第一个 clk_im 来临时，读出的是 0 号单元的指令，你实现了吗?如果没有实现，尝试修改程序实现。(提示: 请关注 rst_n 按键和 clk_im 按键的有效边沿)

答：在复位后，第一次按动 clk_im 按钮,你的程序读出的指令是 0 号单元的。因为先读出所在单元的指令，PC 再进行+4 操作。实现了。

2.你的实验能同时观察 PC 和 IR (即 inst) 的值吗?如果可以，请分析指令地址和指令代码是否匹配?如果不匹配，请分析原因。

答：可以同时观察 PC 和 IR(inst)。指令地址和指令代码匹配。

3.你在板级验证中，每按一下 clk_im 按键,指令是否按序一条一条读出? PC 是否+4 或者只+4?你是如何判断的?分析为什么会出现不正常情况，提出解决办法。

答：是。看 PC 值是否在按下 clk_im 按键+4.无不正常情况。

4. 说说你在实验中碰到了哪些问题，你是如何解决的?

答：编写译码块时测试时出现一直只出现 0 不变的情况，无法分析出正确的立即数。考虑不周全，忘了编写移位指令的逻辑，编写后，可以输出正常结果。