浙江大学

本科实验报告

课程名称：计算机组成

姓　　名：应周骏

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：3200103894

指导教师：马德

2022 年 5 月 14 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：CPU 设计

学生姓名：应周骏 专业：计算机科学与技术 学号：3200103894

同组学生姓名：无 指导老师：马德

实验地点：东 4-509 实验日期：2022 年 3 月 28 日

一、实验目的和要求

1. 复习与运用寄存器传输控制技术；
2. 掌握 CPU 的核心组成：数据通路与控制器；
3. 设计数据通路的功能部件；
4. 进一步了解计算机系统的基本结构；
5. 熟练掌握 IP 核的使用方法；
6. 设计数据通路和控制通路；
7. 学习测试方案的设计；
8. 学习测试程序的设计；

二、实验内容和原理

目标：

熟悉 SOC 系统的原理，掌握 IP 核集成设计 CPU 的方法。

内容：

利用数据通路和控制器两个 IP 核集成设计 CPU（根据原理图采用 RTL 代码方式）。

原理：

1. 单周期 CPU

单周期 CPU 指的是一条指令的执行在一个时钟周期内完成，然后开始下一

条指令的执行，即一条指令用一个时钟周期完成。单周期 CPU，是在一个时钟周期内完成这五个阶段的处理。

(1)取指令(IF)：根据程序计数器 PC 中的指令地址，从存储器中取出一条指令，同时，PC 根据指令字长度自动递增产生下一条指令所需要的指令地址，但遇到“地址转移”指令时，则控制器把“转移地址”送入 PC。

(2)指令译码(ID)：对取指令操作中得到的指令进行分析并译码，确定这条指令需要完成的操作，从而产生相应的操作控制信号，用于驱动执行状态中的各种操作。

(3)指令执行(EXE)：根据指令译码得到的操作控制信号，具体地执行指令动作，然后转移到结果写回状态。

(4)存储器访问(MEM)：所有需要访问存储器的操作都将在这个步骤中执行，该步骤给出存储器的数据地址，把数据写入到存储器中数据地址所指定的存储单元或者从存储器中得到数据地址单元中的数据。

(5)结果写回(WB)：指令执行的结果或者访问存储器中得到的数据写回相应的目的寄存器中。

2. SCPU 模块

通过 SCPU 模块作为连接数据通路和控制通路的中间模块，便于整体嵌入实验 2 中给出的 CPU 测试框架中。

3. 数据通路(Data_path)

CPU 主要部件之一。寄存器传输控制对象为通用数据通路。

3.1 数据通路基本功能

①具有通用计算功能的算术逻辑部件；

②具有通用目的寄存器；

③具有通用计数所需的尽可能的路径；

3.2 架构

①本实验用 IP 软核- Data_path

②核调用模块 Data_path.veo

③核接口信号模块(空文档)：Data_path.v

3.3 具体模块构成：

数据通路作为处理器的一部分，包含了完成处理器所要求的的操作所必须的硬件，包括运算单元、寄存器组、状态寄存器等。

①组合逻辑单元：加法器、多路选择器、ALU 算数运算单元；

②时序逻辑单元（状态元件）：Register、寄存器堆；

4. 控制单元

控制单元作为处理器的一部分，用以告诉数据通路需要做什么。包含了取指单元（PC 及地址计算单元）、译码单元、控制单元（时序信号形成及微操作控制信号形成电路）、中断等。

5. 要求实现的指令

R-Type： add, sub, and, or, xor, slt, sltu,srl, sra, sll；
I-Type： addi, andi, ori, xori, slti, sltiu,srli,srai,slli,lw,jalr;
S-Type： sw；
B-Type： beq,bne；
J-Type： Jal；
U-Type： lui；

图 1 要求实现指令（红色为扩展）

三、实验过程和数据记录

实验 0：SCPU 模块

1. 工程文件建立

新建工程文件，命名为“OxExp04_IP2CPU”。

2. SCPU 模块设计

新建 Verilog 文件“SCPU. v”，输入以下代码。

```
23  module SCPU(
24      input clk,
25      input rst,
26      input MIO_ready,
27      input [31:0]inst_in,
28      input [31:0]Data_in,
29
30      output CPU_MIO,
31      output MemRW,
32      output [31:0]PC_out,
33      output [31:0]Data_out,
34      output [31:0]Addr_out
35      );
36
37      wire [2:0]ALU_Control;
38      wire [1:0]ImmSel;
39      wire [1:0]MemtoReg;
40      wire Jump, Branch, RegWrite, ALUSrc_B;
```

图 3.0.1 SCPU 接口

```
SCPU_ctrl U1(
.OPcode(inst_in[6:2]),
.Fun3(inst_in[14:12]),
.Fun7(inst_in[30]),
.MIO_ready(MIO_ready),
.ImmSel(ImmSel),
.ALUSrc_B(ALUSrc_B),
.MemtoReg(MemtoReg),
.Jump(Jump),
.Branch(Branch),
.RegWrite(RegWrite),
.ALU_Control(ALU_Control),
.CPU_MIO(CPU_MIO)
);

DataPath U0(
    .ALUSrc_B(ALUSrc_B),
    .ALU_Control(ALU_Control),
    .Branch(Branch),
    .Data_in(Data_in),
    .ImmSel(ImmSel),
    .Jump(Jump),
    .MemtoReg(MemtoReg),
    .RegWrite(RegWrite),
    .clk(clk),
    .inst_field(inst_in),
    .rst(rst),
    .ALU_out(Addr_out),
    .Data_out(Data_out),
    .PC_out(PC_out)
);
```

图 3.0.2 连接数据通路和控制通路

3. 调用 IP 模块

加入给出的数据通路、控制通路 IP 模块，整体结构如下。

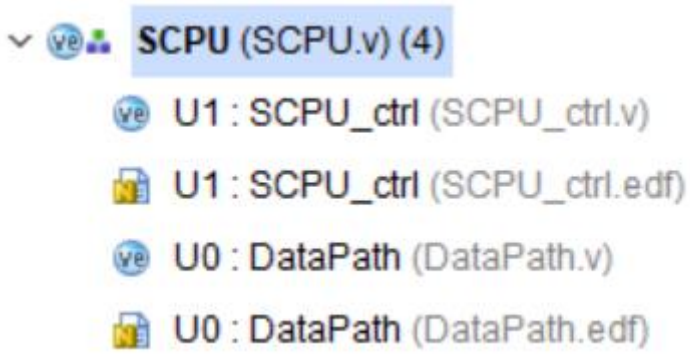

图 3.0.3 模块架构

实验 1：数据通路设计

1. 工程文件建立

在实验 1“OxExp04_SCPU”工程基础上，继续设计控制通路模块，并替换。

2. 数据通路模块

新建 Verilog 文件“Datapath.v”，输入如下代码。

```
module DataPath(
    input wire clk,
    input wire rst,
    input wire[31:0] inst_field,
    input wire[31:0] Data_in,
    input wire[2:0] ALU_operation,
    input wire[1:0] ImmSel,
    input wire[1:0] MemtoReg,
    input wire ALUSrc_B,
    input wire Jump,
    input wire Branch,
    input wire RegWrite,

    output wire[31:0] PC_out,
    output wire[31:0] Data_out,
    output wire[31:0] ALU_out
    );

wire[31:0] Imm_out;
wire[31:0] PC_inc;
wire[31:0] PC_imm;
wire ALU_zero;
wire[31:0] PC_new_1;
wire[31:0] Wt_data;
wire[31:0] PC_new;
wire[31:0] Rs1_data;
wire[31:0] ALU_B;
```

图 3.1.1 数据通路接口

```
ImmGen ImmGen(
    .ImmSel(ImmSel),
    .inst_field(inst_field),
    .Imm_out(Imm_out)
);

add_32 add_32_0(
    .a(PC_out),
    .b(32'h4),
    .c(PC_inc)
);

add_32 add_32_1(
    .a(PC_out),
    .b(Imm_out),
    .c(PC_imm)
);
```

```
MUX2T1_32 MUX2T1_32_1(
    .I0(PC_inc),
    .I1(PC_imm),
    .s(Branch & ALU_zero),
    .O(PC_new_1)
);

MUX4T1_32 MUX4T1_32_0(
    .s(MemtoReg),
    .I0(ALU_out),
    .I1(Data_in),
    .I2(PC_inc),
    .I3(PC_inc),
    .O(Wt_data)
);

MUX2T1_32 MUX2T1_32_3(
    .I0(PC_new_1),
    .I1(PC_imm),
    .s(Jump),
    .O(PC_new)
);

MUX2T1_32 MUX2T1_32_0(
    .I0(Data_out),
    .I1(Imm_out),
    .s(ALUSrc_B),
    .O(ALU_B)
);
```

```
regs Regs(
    .clk(clk),
    .rst(rst),
    .RegWrite(RegWrite),
    .Rs1_addr(inst_field[19:15]),
    .Rs2_addr(inst_field[24:20]),
    .Wt_addr(inst_field[11:7]),
    .Wt_data(Wt_data),
    .Rs1_data(Rs1_data),
    .Rs2_data(Data_out)
);

ALU ALU(
    .A(Rs1_data),
    .B(ALU_B),
    .ALU_operation(ALU_operation),
    .res(ALU_out),
    .zero(ALU_zero)
);

REG32 PC(
    .clk(clk),
    .rst(rst),
    .CE(1'b1),
    .D(PC_new),
    .Q(PC_out)
);

endmodule
```

图 3.1.2 数据通路基本模块连接

3. 设计仿真代码

根据数据通路，设计相应的仿真代码，但仿真未得到预期结果，分析原因，发现是仿真时寄存器值并未随指令执行而改变，导致一次只能测试单条指令，故

输出结果无效，在后续扩展指令设计时，我通过修改 Datapath 模块输出进行测试，由于此处基础指令能通过控制信号仿真和物理验证，故未再单独设计。

```
    .ALU_out(ALU_out)
);

always #10 clk = ~clk;
initial begin

rst = 1'b1;
clk = 1'b0;
inst_field = 32'h00100093;
Data_in = 32'h0;
ImmSel = 2'b00;
MemtoReg = 2'b00;
ALUSrc_B = 1'b0;
ALU_operation = 3'b010;
Jump = 1'b0;
Branch = 1'h0;
RegWrite = 1'b1;
#100;
inst_field = 32'hF80002E3;
Data_in = 32'h0;
ImmSel = 2'b10;
MemtoReg = 2'b00;
ALU_operation = 3'b110;
ALUSrc_B = 1'b0;
Jump = 1'b0;
Branch = 1'h1;
RegWrite = 1'b0;
end
```

图 3.1.3 数据通路仿真代码

实验 2：控制通路设计

1. 工程文件

在实验 1“OxExp04_SCPU”工程基础上，继续设计控制通路模块，并替换。

2. 控制通路模块设计

新建 Verilog 文件“SCPU_ctrl.v”，输入代码。

```
module SCPU_ctrl(
input[4:0]OPcode,                       //Opcode------inst[6:2]
input[2:0]Fun3,                          //Function-----inst[14:12]
input Fun7,                                 //Function-----inst[30]
input MIO_ready,                        //CPU Wait
output reg [1:0]ImmSel,
output reg ALUSrc_B,
output reg [1:0]MemtoReg,
output reg Jump,                        //jal
output reg Branch,                      //beq
output reg RegWrite,
output reg MemRW,
output reg [2:0]ALU_Control,       //alu
output reg CPU_MIO                    //not use
);

reg [1:0] ALUop;
wire [3:0] Fun;
reg CPU_ctrl_signals;
`define CPU_ctrl_signals {ALUSrc_B,MemtoReg,RegWrite,MemRW,Branch,Jump,ALUop,ImmSel}
```

图 3.2.1 控制通路基本接口

```
always @* begin
case(OPcode)
 5'b01100: begin {ALUSrc_B,MemtoReg,RegWrite,MemRW,Branch,Jump,ALUop,ImmSel} = 11'b0_00_1_0_0_0_10_00; end //ALU
 5'b00000: begin {ALUSrc_B,MemtoReg,RegWrite,MemRW,Branch,Jump,ALUop,ImmSel} = 11'b1_01_1_0_0_0_00_00; end //load
 5'b01000: begin {ALUSrc_B,MemtoReg,RegWrite,MemRW,Branch,Jump,ALUop,ImmSel} = 11'b1_00_0_1_0_0_00_01; end //store
 5'b11000: begin {ALUSrc_B,MemtoReg,RegWrite,MemRW,Branch,Jump,ALUop,ImmSel} = 11'b0_00_0_0_1_0_01_10; end //beq
 5'b11011: begin {ALUSrc_B,MemtoReg,RegWrite,MemRW,Branch,Jump,ALUop,ImmSel} = 11'b0_10_1_0_0_1_00_11; end //jump
 5'b00100: begin {ALUSrc_B,MemtoReg,RegWrite,MemRW,Branch,Jump,ALUop,ImmSel} = 11'b1_00_1_0_0_0_11_00; end //ALU(addi;;;;)
 default: begin {ALUSrc_B,MemtoReg,RegWrite,MemRW,Branch,Jump,ALUop,ImmSel} = 11'b00000000000; end
endcase
end
```

图 3.2.2 控制通路基本信号生成

```
assign Fun = {Fun3,Fun7};

always @* begin
case(ALUop)
 2'b00: ALU_Control = 3'b010 ; //add
 2'b01: ALU_Control = 4'b110 ; //sub
 2'b10:
case(Fun)
 4'b0000: ALU_Control = 3'b010 ; //add
 4'b0001: ALU_Control = 3'b110 ; //sub
 4'b1110: ALU_Control = 3'b000 ; //and
 4'b1100: ALU_Control = 3'b001 ; //or
 4'b0100: ALU_Control = 3'b111 ; //slt
 4'b1010: ALU_Control = 3'b101 ; //srl
 4'b1000: ALU_Control = 3'b011 ; //xor
 default: ALU_Control = 3'bx;  //nor(no this kind)
endcase
 2'b11:
case(Fun3)
 3'b000: ALU_Control = 3'b010 ; //addi
 3'b010: ALU_Control = 3'b111 ; //slti
 3'b100: ALU_Control = 3'b011 ; //xori
 3'b110: ALU_Control = 3'b001 ; //ori
 3'b111: ALU_Control = 3'b000 ; //andi
 3'b101: ALU_Control = 3'b101 ; //srli
 default: ALU_Control = 3'bx ;  //nor(no this kind)
endcase
endcase
end
endmodule
```

图 3.2.3 控制通路 ALU 两级译码

3. 控制通路仿真

对该原理图进行仿真，输入如下仿真代码。

```
module SCPU_tb(
);
reg[4:0] OPcode;
reg MIO_ready;
reg [2:0]Fun3;
reg Fun7;
reg Fun;
wire [1:0]ImmSel;
wire ALUSrc_B;
wire Jump;
wire [1:0] MemtoReg;
wire Branch;
wire RegWrite;
wire [2:0]ALU_Control;
wire CPU_MIO;


SCPU_ctrl U1(
    .OPcode(OPcode),
    .Fun3(Fun3),
    .Fun7(Fun7),
    .MIO_ready(MIO_ready),
    .ImmSel(ImmSel),
    .ALUSrc_B(ALUSrc_B),
    .MemtoReg(MemtoReg),
    .Jump(Jump),
    .Branch(Branch),
    .RegWrite(RegWrite),
    .ALU_Control(ALU_Control),
    .CPU_MIO(CPU_MIO)
    );
```

```
OPcode = 5'b01100; //ALU指令,检查 ALUop=2'b10; RegWrite=1
Fun3 = 3'b000; Fun7 = 1'b0;//add,检查ALU_Control=3'b010
#20;
Fun3 = 3'b000; Fun7 = 1'b1;//sub,检查ALU_Control=3'b110
#20;
Fun3 = 3'b111; Fun7 = 1'b0;//and,检查ALU_Control=3'b000
#20;
Fun3 = 3'b110; Fun7 = 1'b0;//or,检查ALU_Control=3'b001
#20;
Fun3 = 3'b010; Fun7 = 1'b0 ;//slt,检查ALU_Control=3'b111
#20;
#20;
Fun3 = 3'b101; Fun7 = 1'b0; //srl,检查ALU_Control=3'b101
#20;
Fun3 = 3'b100; Fun7 = 1'b0; //xor,检查ALU_Control=3'b011
#20;
Fun3 = 3'b111; Fun7 = 1'b1; //间隔
#20;
OPcode = 5'b00000; //load指令,检查 ALUop=2'b00, #20; // ALUSrc_B=1, MemtoReg=1, RegWrite=1
OPcode = 5'b01000; #20; //store指令,检查ALUop=2'b00, MemRW=1, ALUSrc_B=1
OPcode = 5'b11000; #20//beq指令,检查 ALUop=2'b01, Branch=1 #20;
OPcode = 5'b11011; #20 //jump指令,检查 Jump=1
OPcode = 5'b00100; #20 //I指令,检查 ALUop=2'b11; RegWrite=1 #20;
Fun3 = 3'b000; //addi,检查ALU_Control=3'b010
#20;
OPcode = 5'h1f; //间隔
Fun3 = 3'b000; Fun7 = 1'b0;//间隔
end
```

图 3.2.4 控制通路仿真代码

得到仿真结果。

图 3.2.5 控制通路仿真结果图

4. 立即数生成模块设计

基于控制通路信号，设计对应的立即数生成模块 Imm_Gen，新建 Verilog 文件“Imm_Gen.v”，输入如下代码。

```
module ImmGen(
    input wire [1:0] ImmSel,
    input wire [31:0] inst_field,
    output reg [31:0] Imm_out
    );

    always@*begin
    case(ImmSel)
    2'b00:Imm_out = {{20{inst_field[31]}},inst_field[31:20]}; //addi\lw(I)
    2'b01:Imm_out = {{20{inst_field[31]}},inst_field[31:25],inst_field[11:7]}; //sw(s)
    2'b10:Imm_out = {{20{inst_field[31]}},inst_field[7],inst_field[30:25],inst_field[11:8],1'b0}; //beq(b)
    2'b11:Imm_out = {{12{inst_field[31]}},inst_field[19:12],inst_field[20],inst_field[30:21],1'b0}; //jal(j)
    endcase
    end
    endmodule
```

图 3.2.6 立即数生成器

5. PC 寄存器

根据数据通路，设计 PC 寄存器，新建文件“REG32.v”，输入如下代码。

```
module REG32(
    input clk,
    input rst,
    input CE,
    input [31:0]D,
    output reg [31:0]Q
    );

always @(posedge clk or posedge rst)
if (rst==1) Q <= 32'b0;
else if (CE == 1) Q <= D;
endmodule
```

图 3.2.7 32 位 PC 寄存器模块

6. 集成到实验 2 的 CPU 测试环境

将设计的 SCPU、数据通路模块、控制通路模块整合到实验 2 搭建的测试模块上，整体架构如下。

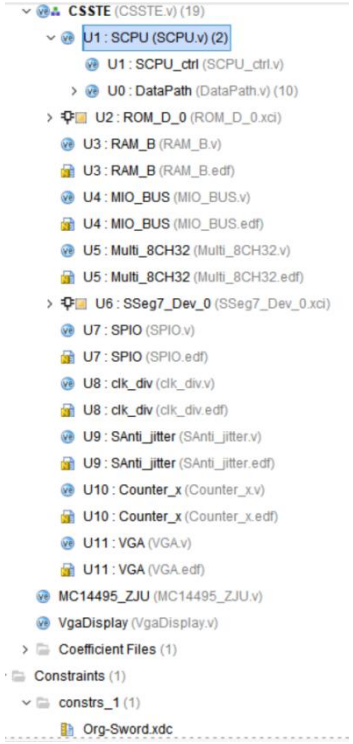

图 3.2.8 测试框架架构

对该工程生成对应的 bit 文件，在 SWORD 板上进行验证。

实验 3：扩展指令设计

1. 实验工程修改

在前面基础指令实验的基础上，新建工程“OxExp04-IP2CPU_extension”，更改数据通路和控制通路，适应扩展指令集要求。

2. 数据通路模块调整

修改基础指令的数据通路，拓展二级的跳转指令 MUX 的位数，并更改部分通路连接，输入如下代码。

```
module DataPath_more(
    input wire clk,
    input wire rst,
    input wire[31:0] inst_field,
    input wire[31:0] Data_in,
    input wire[3:0] ALU_operation,
    input wire[2:0] ImmSel,
    input wire[1:0] MemtoReg,
    input wire ALUSrc_B,
    input wire[1:0]Jump,
    input wire Branch,
    input wire BranchN,
    input wire RegWrite,

    output wire[31:0] PC_out,
    output wire[31:0] Data_out,
    output wire[31:0] ALU_out
    );

wire[31:0] Imm_out;
wire[31:0] PC_inc;
wire[31:0] PC_imm;
wire ALU_zero;
wire[31:0] PC_new_1;
wire[31:0] Wt_data;
wire[31:0] PC_new;
wire[31:0] Rs1_data;
wire[31:0] ALU_B;
```

图 3.3.1 扩展指令数据通路接口

```
ImmGen ImmGen(
    .ImmSel(ImmSel),
    .inst_field(inst_field),
    .Imm_out(Imm_out)
);

add_32 add_32_0(
    .a(PC_out),
    .b(32'h4),
    .c(PC_inc)
);

add_32 add_32_1(
    .a(PC_out),
    .b(Imm_out),
    .c(PC_imm)
);

MUX2T1_32 MUX2T1_32_1(
    .I0(PC_inc),
    .I1(PC_imm),
    .s((Branch & ALU_zero) | (BranchN & ~ALU_zero)),
    .O(PC_new_1)
);

MUX4T1_32 MUX4T1_32_0(
    .s(MemtoReg),
    .I0(ALU_out),
    .I1(Data_in),
    .I2(PC_inc),
    .I3(Imm_out),
    .O(Wt_data)
);

MUX4T1_32 MUX4T1_32_1(
    .s(Jump),
    .I0(PC_new_1),
    .I1(PC_imm),
    .I2(ALU_out),
    .I3(PC_new_1),
    .O(PC_new)
);

MUX2T1_32 MUX2T1_32_0(
    .I0(Data_out),
    .I1(Imm_out),
    .s(ALUSrc_B),
    .O(ALU_B)
);

regs Regs(
    .clk(clk),
    .rst(rst),
    .RegWrite(RegWrite),
    .Rs1_addr(inst_field[19:15]),
    .Rs2_addr(inst_field[24:20]),
    .Wt_addr(inst_field[11:7]),
    .Wt_data(Wt_data),
    .Rs1_data(Rs1_data),
    .Rs2_data(Data_out)
);

ALU ALU(
    .A(Rs1_data),
    .B(ALU_B),
    .ALU_operation(ALU_operation),
    .res(ALU_out),
    .zero(ALU_zero)
);

REG32 PC(
    .clk(clk),
    .rst(rst),
    .CE(1'b1),
    .D(PC_new),
    .Q(PC_out)
);
```

图 3.3.2 扩展指令数据通路连接

3. 控制通路模块修改

对与新增指令，修改对应的控制信号，支持 jalr,lui,bne 等指令要求。

```
module SCPU_ctrl_more(
input[4:0]OPcode,                       //Opcode------inst[6:2]
input[2:0]Fun3,                          //Function-----inst[14:12]
input Fun7,                                 //Function-----inst[30]
input MIO_ready,                        //CPU Wait
output reg [2:0]ImmSel,
output reg ALUSrc_B,
output reg [1:0]MemtoReg,
output reg [1:0]Jump,                       //jal
output reg Branch,                      //beq
output reg BranchN,
output reg RegWrite,
output reg MemRW,
output reg [3:0]ALU_Control,        //alu
output reg CPU_MIO                      //not use
);

reg [1:0] ALUop;
wire [3:0] Fun;
```

图 3.3.3 扩展指令控制通路接口

```
always @* begin
case(OPcode)
5'b01100: begin {ALUSrc_B,MemtoReg,RegWrite,MemRW,Branch,BranchN,Jump,ALUop,ImmSel}  = 14'b0_00_1_0_0_0_00_10_000; end //ALU
5'b00000: begin {ALUSrc_B,MemtoReg,RegWrite,MemRW,Branch,BranchN,Jump,ALUop,ImmSel}  = 14'b1_01_1_0_0_0_00_00_001; end //load
5'b01000: begin {ALUSrc_B,MemtoReg,RegWrite,MemRW,Branch,BranchN,Jump,ALUop,ImmSel}  = 14'b1_00_0_1_0_0_00_00_010; end //store
5'b11000: begin
case(Fun3)
3'b000: begin {ALUSrc_B,MemtoReg,RegWrite,MemRW,Branch,BranchN,Jump,ALUop,ImmSel}  = 14'b0_00_0_0_1_0_00_01_011; end
3'b001: begin {ALUSrc_B,MemtoReg,RegWrite,MemRW,Branch,BranchN,Jump,ALUop,ImmSel}  = 14'b0_00_0_0_0_1_00_01_011; end
endcase //beq
end
5'b11011: begin {ALUSrc_B,MemtoReg,RegWrite,MemRW,Branch,BranchN,Jump,ALUop,ImmSel}  = 14'b0_10_1_0_0_0_01_00_101; end //jump
5'b00100: begin {ALUSrc_B,MemtoReg,RegWrite,MemRW,Branch,BranchN,Jump,ALUop,ImmSel}  = 14'b1_00_1_0_0_0_00_11_001; end //ALU(addi;;;;)
5'b11001: begin {ALUSrc_B,MemtoReg,RegWrite,MemRW,Branch,BranchN,Jump,ALUop,ImmSel}  = 14'b1_10_1_0_0_0_10_00_001; end //jarl
5'b01101: begin {ALUSrc_B,MemtoReg,RegWrite,MemRW,Branch,BranchN,Jump,ALUop,ImmSel}  = 14'b0_11_1_0_0_0_00_00_000; end //lui
default: begin  {ALUSrc_B,MemtoReg,RegWrite,MemRW,Branch,BranchN,Jump,ALUop,ImmSel}  = 14'h0; end
endcase
end
```

图 3.3.4 扩展指令控制通路

```
assign Fun = {Fun3,Fun7};

always @* begin
case(ALUop)
2'b00: ALU_Control = 4'b0010 ; //add
2'b01: ALU_Control = 4'b0110 ; //sub
2'b10:
case(Fun)
4'b0000: ALU_Control = 4'b0010 ; //add
4'b0001: ALU_Control = 4'b0110 ; //sub
4'b1110: ALU_Control = 4'b0000 ; //and
4'b1100: ALU_Control = 4'b0001 ; //or
4'b0100: ALU_Control = 4'b0111 ; //slt
4'b1010: ALU_Control = 4'b0101 ; //srl
4'b1000: ALU_Control = 4'b0011 ; //xor
4'b0110: ALU_Control = 4'b1010 ; //sltu
4'b0010: ALU_Control = 4'b1001 ; //sll
4'b1011: ALU_Control = 4'b1000 ; //sra
default: ALU_Control = 4'bx;  //nor(no this kind)
endcase
2'b11:
case(Fun3)
3'b000: ALU_Control = 4'b0010 ; //addi
3'b010: ALU_Control = 4'b0111 ; //slti
3'b100: ALU_Control = 4'b0011 ; //xori
3'b110: ALU_Control = 4'b0001 ; //ori
3'b111: ALU_Control = 4'b0000 ; //andi
3'b101:
    if(Fun7)
        ALU_Control = 4'b1000 ; //srai
    else
        ALU_Control = 4'b0101 ; //srli
3'b011: ALU_Control = 4'b1010 ; //sltiu
3'b001: ALU_Control = 4'b1001 ; //slli
default: ALU_Control = 4'bx ;  //nor(no this kind)
endcase
endcase
end
endmodule
```

图 3.3.5 扩展指令控制通路 ALU 二级译码

4. 立即数生成模块调整

根据控制通路的变化，修改立即数生成模块，适应 lui 指令的要求。

```
module ImmGen(
    input wire [2:0] ImmSel,
    input wire [31:0] inst_field,
    output reg [31:0] Imm_out
    );

    always@*begin
    case(ImmSel)
    3'b001:Imm_out = {{20{inst_field[31]}},inst_field[31:20]}; //addi\lw(I)
    3'b010:Imm_out = {{20{inst_field[31]}},inst_field[31:25],inst_field[11:7]}; //sw(s)
    3'b011:Imm_out = {{20{inst_field[31]}},inst_field[7],inst_field[30:25],inst_field[11:8],1'b0}; //beq(b)
    3'b101:Imm_out = {{12{inst_field[31]}},inst_field[19:12],inst_field[20],inst_field[30:21],1'b0}; //jal(j)
    3'b000:Imm_out = {inst_field[31:12],12'h0}; //U-type/lui
    default:Imm_out = 32'h4;
    endcase
    end
    endmodule
```

图 3.3.6 扩展指令立即数模块

5. ALU 模块调整

由于扩展指令拓展了左移、有符号数比较等运算，故对 ALU 模块进行相应的修改，并扩展控制信号位数。

```
module ALU(
    input [31:0] A,
    input [3:0] ALU_operation,
    input [31:0] B,
    output reg [31:0] res,
    output zero
);
parameter one = 32'h00000001, zero_0 = 32'h00000000;
always @ (*)
    case (ALU_operation)
    4'b0000: res = A&B; //and
    4'b0001: res = A|B; //or
    4'b0010: res = A+B; //add
    4'b0011: res = A^B; //xor
    4'b0100: res = ~(A|B); //nor
    4'b0101: res = A>>B;  //srl
    4'b0110: res = A-B; //sub
    4'b0111: res = ($signed(A)<$signed(B))?one:zero_0; //slt
    4'b1000: res = ($signed(A) >>> B); //sra
    4'b1001: res = A<<B; //sll
    4'b1010: res = (A<B)?one:zero_0; //sltu
    default: res = 32'hx; //
    endcase

    assign zero = (res==0)? 1: 0;
endmodule
```

图 3.3.7 ALU 模块拓展

6. 仿真设计

针对数据通路和控制通路进行仿真，其中，对数据通路，主要测试立即数生成，故修改了部分输出值，便于查看仿真结果。

```
reg clk;
reg rst;
reg[31:0] inst_field;
reg[31:0] Data_in;
reg[3:0] ALU_operation;
reg[2:0] ImmSel;
reg[1:0] MemtoReg;
reg ALUSrc_B;
reg[1:0]Jump;
reg Branch;
reg BranchN;
reg RegWrite;

wire[31:0] PC_out;
wire[31:0] Data_out;
wire[31:0] Imm_out;
wire[31:0] PC_new;
DataPath_more U1(
    .clk(clk),
    .rst(rst),
    .inst_field(inst_field),
    .Data_in(Data_in),
    .ImmSel(ImmSel),
    .MemtoReg(MemtoReg),
    .ALUSrc_B(ALUSrc_B),
    .Jump(Jump),
    .Branch(Branch),
    .BranchN(BranchN),
    .RegWrite(RegWrite)
```

```
rst = 1'b1;
clk = 1'b0;
inst_field = 32'h88888137; //lui
Data_in = 32'h80000000;
ImmSel = 3'b000;
MemtoReg = 2'b11;
ALUSrc_B = 1'b0;
Jump = 2'b00;
Branch = 1'h0;
BranchN = 1'h0;
RegWrite = 1'b1;
ALU_operation = 4'b0010;
#100;
inst_field = 32'hFE0098E3; //bne
Data_in = 32'h00000000;
ImmSel = 3'b011;
MemtoReg = 2'b00;
ALUSrc_B = 1'b0;
Jump = 2'b00;
Branch = 1'h0;
BranchN = 1'h1;
RegWrite = 1'b0;
ALU_operation = 4'b0110;
end
```

图 3.3.8 数据通路仿真

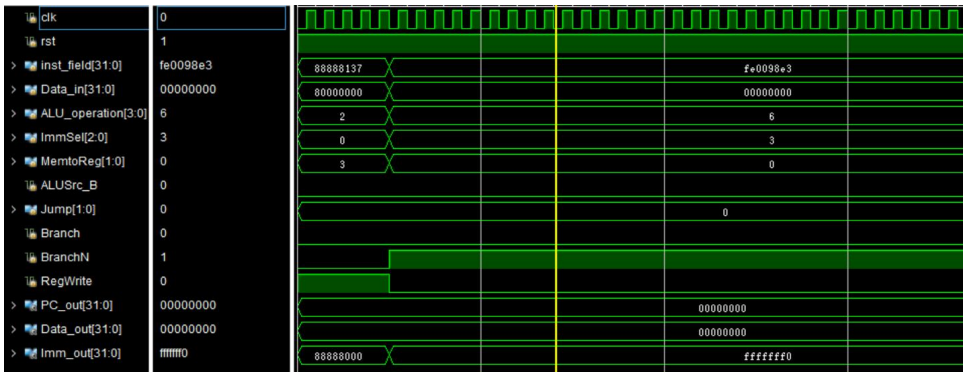

图 3.3.9 数据通路仿真结果

```
initial begin
    // Initialize Inputs
    OPcode = 0;
    Fun3 = 0;
    Fun7 = 0;
    MIO_ready = 0; #40;
    // Wait 40 ns for global reset to finish
    // Add stimulus here
    OPcode = 5'b11001;
    #20;
    OPcode = 5'b01101;
    #20;
    OPcode = 5'b01100; // ALUop=2'b10; RegWrite=1
    Fun3 = 3'b000; Fun7 = 1'b0;//add,ALU_Control=3'b010
    #20;
    Fun3 = 3'b000; Fun7 = 1'b1;//sub,ALU_Control=3'b110
    #20;
    Fun3 = 3'b111; Fun7 = 1'b0;//and,ALU_Control=3'b000
    #20;
    Fun3 = 3'b110; Fun7 = 1'b0;//or,ALU_Control=3'b001
    #20;
    Fun3 = 3'b010; Fun7 = 1'b0 ;//slt,ALU_Control=3'b111
    #20;
    Fun3 = 3'b101; Fun7 = 1'b0; //srl,ALU_Control=3'b101
    #20;
    Fun3 = 3'b100; Fun7 = 1'b0; //xor,ALU_Control=3'b011
    #20;
    Fun3 = 3'b001; Fun7 = 1'b0; //sll
    #20
    Fun3 = 3'b011; Fun7 = 1'b0; //sltu
    #20
    Fun3 = 3'b101; Fun7 = 1'b1; //sra
    #20
    Fun3 = 3'b111; Fun7 = 1'b1; //
    #20;
    OPcode = 5'b00000; //load, ALUop=2'b00,
    #20; // ALUSrc_B=1, MemtoReg=1, RegWrite=1
    OPcode = 5'b01000;
    #20; //store,ALUop=2'b00, MemRW=1, ALUSrc_B=1
    OPcode = 5'b11000;//beq,ALUop=2'b01, Branch=1
    Fun3 = 3'b000;
    #20
    Fun3 = 3'b001;
    #20;
    OPcode = 5'b11011; //jump Jump=1
    #40;
    OPcode = 5'b00100; //I指, ALUop=2'b11; RegWrite=1
    Fun3 = 3'b000;  //addi,ALU_Control=3'b010
    #20;
    Fun3 = 3'b111;//andi
    #20;
    Fun3 = 3'b110; //ori
    #20;
    Fun3 = 3'b010; //slti
    #20;
    Fun3 = 3'b101; //srli
    Fun7 = 0;
    #20
    Fun3 = 3'b101; //srai
    Fun7 = 1;
    #20;
    Fun3 = 3'b100; //xori
    #20;
    Fun3 = 3'b011; //sltiu
    #20
    Fun3 = 3'b001; //slli
    #20
    Fun3 = 3'b100; //xori
    #20
    OPcode = 5'h1f;
    Fun3 = 3'b000; Fun7 = 1'b0;
    end
endmodule
```

图 3.3.10 控制通路仿真代码

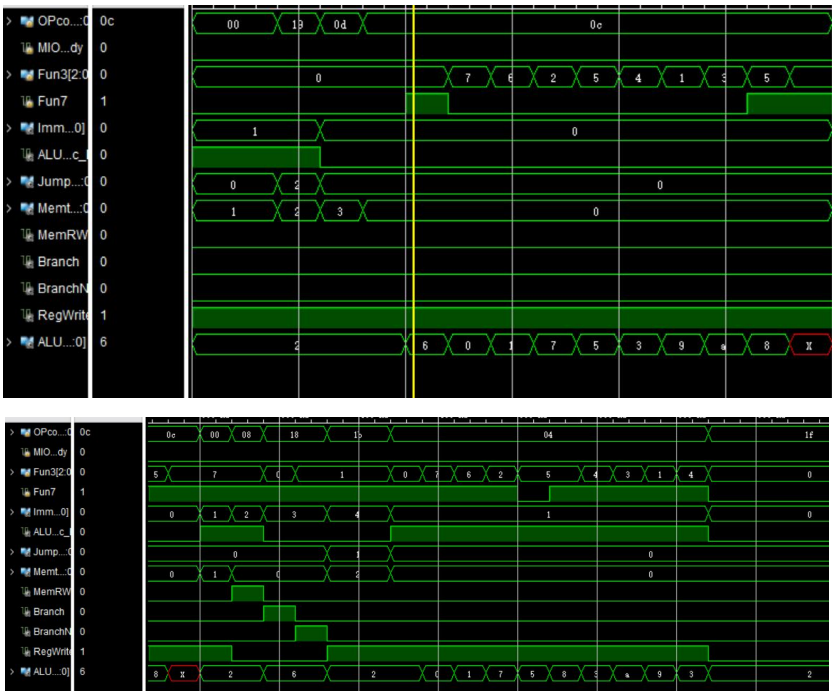

图 3.3.11 控制通路仿真结果

7. 物理验证

对上述扩展指令生成 bit 文件，在 SWORD 实验板上进行验证。

四、实验结果分析

基础指令集单周期 CPU：

1. 数据通路代码分析

数据通路的实现本质上是对各个模块的连线，此处不做赘述，下面是各个模块的示意图。

图 4.1.1 数据通路模块构成

2. 控制通路代码及仿真分析

控制通路代码的设计核心在于对不同的操作码进行控制信号的确定，并将 ALU 信号通过两级解码，进行最终的确定。

```
always @* begin
case(OPcode)
 5'b01100: begin {ALUSrc_B,MemtoReg,RegWrite,MemRW,Branch,Jump,ALUop,ImmSel} = 11'b0_00_1_0_0_0_10_00; end //ALU
 5'b00000: begin {ALUSrc_B,MemtoReg,RegWrite,MemRW,Branch,Jump,ALUop,ImmSel} = 11'b1_01_1_0_0_0_00_00; end //load
 5'b01000: begin {ALUSrc_B,MemtoReg,RegWrite,MemRW,Branch,Jump,ALUop,ImmSel} = 11'b1_00_0_1_0_0_00_01; end //store
 5'b11000: begin {ALUSrc_B,MemtoReg,RegWrite,MemRW,Branch,Jump,ALUop,ImmSel} = 11'b0_00_0_0_1_0_01_10; end //beq
 5'b11011: begin {ALUSrc_B,MemtoReg,RegWrite,MemRW,Branch,Jump,ALUop,ImmSel} = 11'b0_10_1_0_0_1_00_11; end //jump
 5'b00100: begin {ALUSrc_B,MemtoReg,RegWrite,MemRW,Branch,Jump,ALUop,ImmSel} = 11'b1_00_1_0_0_0_11_00; end //ALU(addi;;;;)
 default: begin {ALUSrc_B,MemtoReg,RegWrite,MemRW,Branch,Jump,ALUop,ImmSel} = 11'b00000000000; end
endcase
end
```

各个控制信号　　对信号赋值

图 4.1.2 控制通路基本信号

```
assign Fun = {Fun3,Fun7};

always @* begin                                  一级解码
case(ALUop)                                      jal,lw,sd
 2'b00: ALU_Control = 3'b010 ; //add             beq
 2'b01: ALU_Control = 4'b110 ; //sub
 2'b10:
case(Fun)
 4'b0000: ALU_Control = 3'b010 ; //add
 4'b0001: ALU_Control = 3'b110 ; //sub           二级
 4'b1110: ALU_Control = 3'b000 ; //and
 4'b1100: ALU_Control = 3'b001 ; //or
 4'b0100: ALU_Control = 3'b111 ; //slt
 4'b1010: ALU_Control = 3'b101 ; //srl
 4'b1000: ALU_Control = 3'b011 ; //xor
 default: ALU_Control = 3'bx;  //nor(no this kind)
endcase                                          ALU运算操作
 2'b11:                                          立即数
case(Fun3)
 3'b000: ALU_Control = 3'b010 ; //addi
 3'b010: ALU_Control = 3'b111 ; //slti
 3'b100: ALU_Control = 3'b011 ; //xori
 3'b110: ALU_Control = 3'b001 ; //ori
 3'b111: ALU_Control = 3'b000 ; //andi
 3'b101: ALU_Control = 3'b101 ; //srli
 default: ALU_Control = 3'bx ;  //nor(no this kind)
endcase
endcase
end
endmodule
```

图 4.1.3 控制通路 ALU 信号

仿真主要针对不同操作，观察输出的控制信号是否符合要求。

图 4.1.4 控制通路仿真结果分析

3. 物理验证结果

测试程序为斐波那契数列，运行能够显示正确结果，ALU_res 和预期一致。

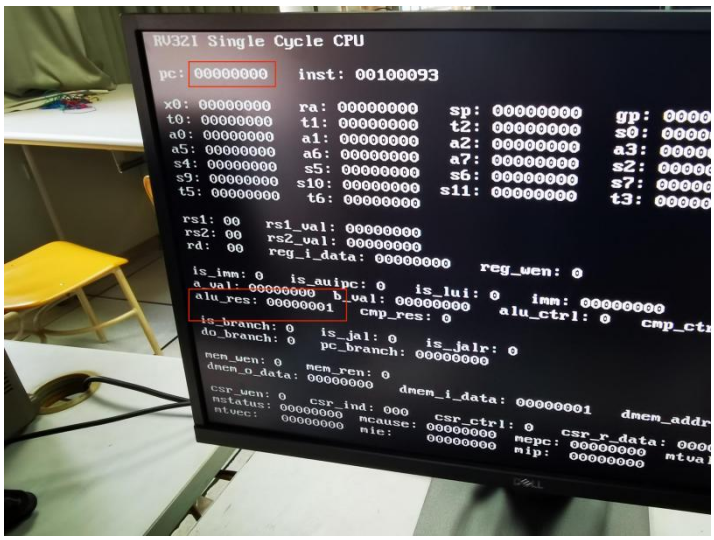

图 4.1.5 起始状态 ALU 结果

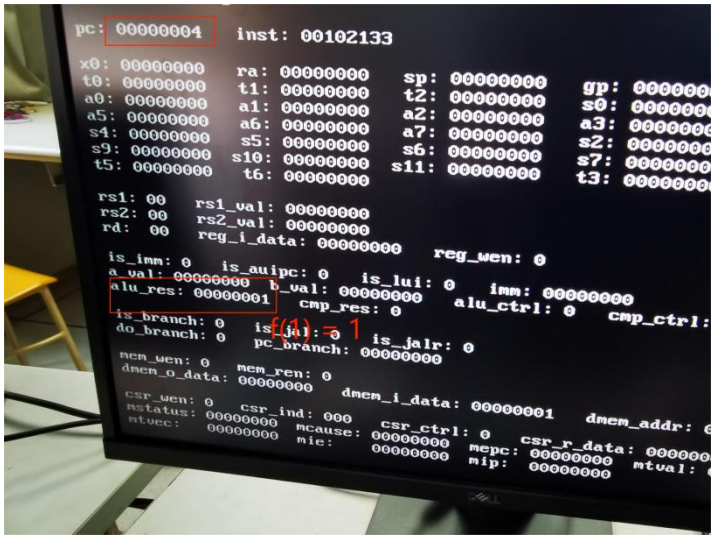

图 4.1.6 下一步指令(PC = 4)

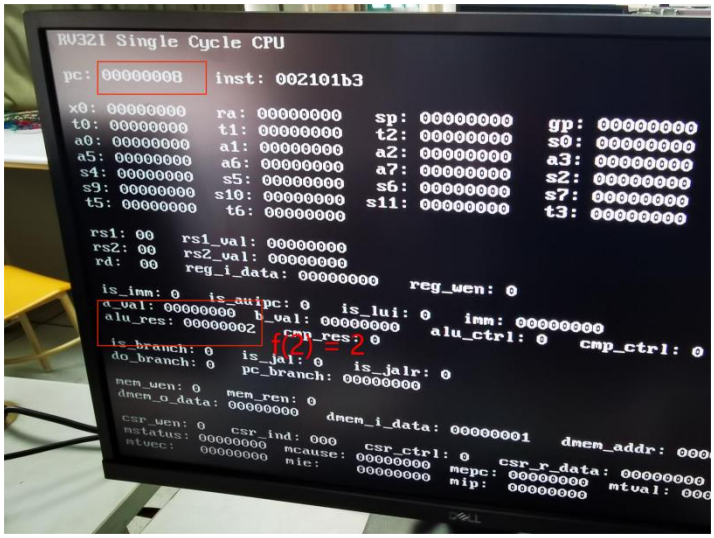

图 4.1.7 PC=8

扩展指令集单周期 CPU：

1. 数据通路更改

数据通路的修改主要在于对 PC 更新的模块进行扩展，由于需要实现 bne，故拓展了原本两位的多路选择器，以实现功能。

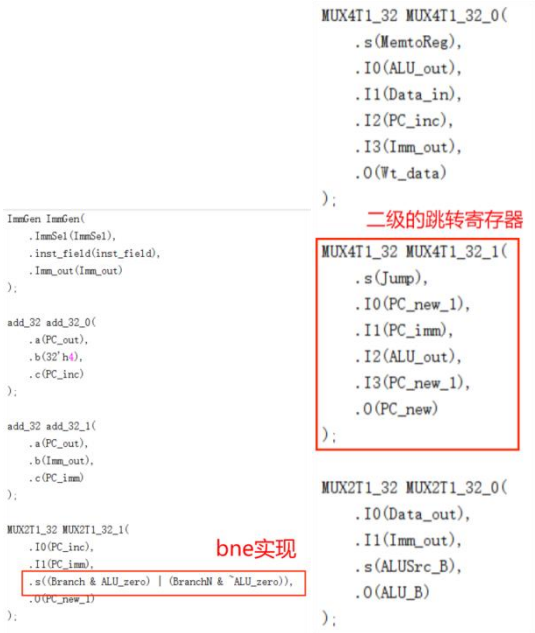

图 4.2.1 数据通路扩展

2. 控制通路更改

由于需要增加实现 jarl 指令，虽然是 I 型，但是跳转操作和一般的立即数运算操作有不同，且与 jal 指令控制信号也有区别，故新增了对 jarl 指令的信号生成语句。类似地，lui 指令为 U 型指令，额外编写信号生成语句，并将 B 型指令

根据 Fun3 细分为 bne 和 beq，进行相应调整。在 ALU 二级译码过程中，主要增加需要支持的 sll、sra、slt 指令以及其相应的立即数版本指令。在修改数据通路时，也相应修改立即数生成模块。

```
always @* begin
case(OPcode)
5'b01100: begin {ALUSrc_B,MemtoReg,RegWrite,MemRW,Branch,BranchN,Jump,ALUop,ImmSel}  = 14'b0_00_1_0_0_0_00_10_000; end //ALU
5'b00000: begin {ALUSrc_B,MemtoReg,RegWrite,MemRW,Branch,BranchN,Jump,ALUop,ImmSel}  = 14'b1_01_1_0_0_0_00_00_001; end //load
5'b01000: begin {ALUSrc_B,MemtoReg,RegWrite,MemRW,Branch,BranchN,Jump,ALUop,ImmSel}  = 14'b1_00_0_1_0_0_00_00_010; end //store
5'b11000: begin
case(Fun3)
3'b000: begin {ALUSrc_B,MemtoReg,RegWrite,MemRW,Branch,BranchN,Jump,ALUop,ImmSel}  = 14'b0_00_0_0_1_0_00_01_011; end
3'b001: begin {ALUSrc_B,MemtoReg,RegWrite,MemRW,Branch,BranchN,Jump,ALUop,ImmSel}  = 14'b0_00_0_0_0_1_00_01_011; end
endcase //beq
end
5'b11011: begin {ALUSrc_B,MemtoReg,RegWrite,MemRW,Branch,BranchN,Jump,ALUop,ImmSel}  = 14'b0_10_1_0_0_0_01_00_101; end //jump
5'b00100: begin {ALUSrc_B,MemtoReg,RegWrite,MemRW,Branch,BranchN,Jump,ALUop,ImmSel}  = 14'b1_00_1_0_0_0_00_11_001; end //ALU(addi;;;;)
5'b11001: begin {ALUSrc_B,MemtoReg,RegWrite,MemRW,Branch,BranchN,Jump,ALUop,ImmSel}  = 14'b1_10_1_0_0_0_10_00_001; end //jarl
5'b01101: begin {ALUSrc_B,MemtoReg,RegWrite,MemRW,Branch,BranchN,Jump,ALUop,ImmSel}  = 14'b0_11_1_0_0_0_00_00_000; end //lui
default: begin  {ALUSrc_B,MemtoReg,RegWrite,MemRW,Branch,BranchN,Jump,ALUop,ImmSel}  = 14'h0; end
endcase
end
```

细分B型指令

图 4.2.2 扩展指令控制信号生成模块

```
assign Fun = {Fun3,Fun7};

always @* begin
case(ALUop)
2'b00: ALU_Control = 4'b0010 ; //add
2'b01: ALU_Control = 4'b0110 ; //sub
2'b10:
case(Fun)
4'b0000: ALU_Control = 4'b0010 ; //add
4'b0001: ALU_Control = 4'b0110 ; //sub
4'b1110: ALU_Control = 4'b0000 ; //and
4'b1100: ALU_Control = 4'b0001 ; //or
4'b0100: ALU_Control = 4'b0111 ; //slt
4'b1010: ALU_Control = 4'b0101 ; //srl
4'b1000: ALU_Control = 4'b0011 ; //xor
4'b0110: ALU_Control = 4'b1010 ; //sltu
4'b0010: ALU_Control = 4'b1001 ; //sll
4'b1011: ALU_Control = 4'b1000 ; //sra
default: ALU_Control = 4'bx;  //nor(no this kind)
endcase
2'b11:
case(Fun3)
3'b000: ALU_Control = 4'b0010 ; //addi
3'b010: ALU_Control = 4'b0111 ; //slti
3'b100: ALU_Control = 4'b0011 ; //xori
3'b110: ALU_Control = 4'b0001 ; //ori
3'b111: ALU_Control = 4'b0000 ; //andi
3'b101:
    if(Fun7)
        ALU_Control = 4'b1000 ; //srai
    else
        ALU_Control = 4'b0101 ; //srli
3'b011: ALU_Control = 4'b1010 ; //sltiu
3'b001: ALU_Control = 4'b1001 ; //slli
default: ALU_Control = 4'bx ;  //nor(no this kind)
endcase
endcase
end
endmodule
```

图 4.2.3 扩展指令 ALU 二级解码

3. ALU 模块更改

实验需要实现运算指令的拓展，则超出了原本三位的 ALU 信号能表示的操作范围，故扩展控制信号为 4 位，并实现相应的 ALU 操作，同时注意操作数是有符号数还是无符号数。

4. 仿真结果分析

对数据通路的仿真主要测试立即数是否生成成功，这是由于我在实验时遇到了立即数的生成错误问题。由图中可以看到，第一条测试的指令为 lui x2, 0x88888，可以看到生成的立即数为 88888000，符合 lui 立即数生成要求；第二条为 bne x1 x0 -16，生成立即数为 fffffff0，转化为十进制即为-16，符合要求。

对控制通路的仿真与基础指令集类似，增加了对扩展指令的仿真验证，此处不赘述。

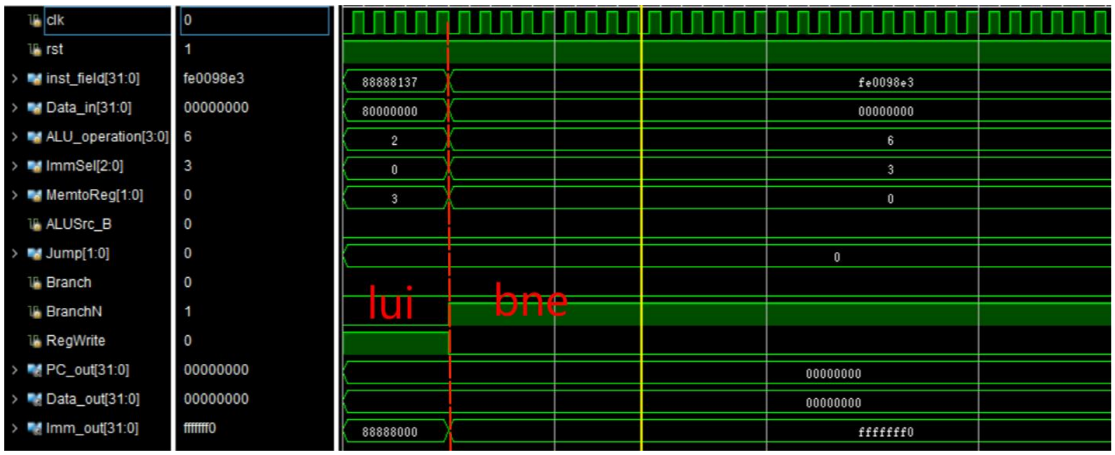

图 4.2.4 数据通路仿真结果分析

5. 物理验证结果

根据测试代码逐步检查 ALU、memory 结果，符合实验预期。具体结果见附件视频。

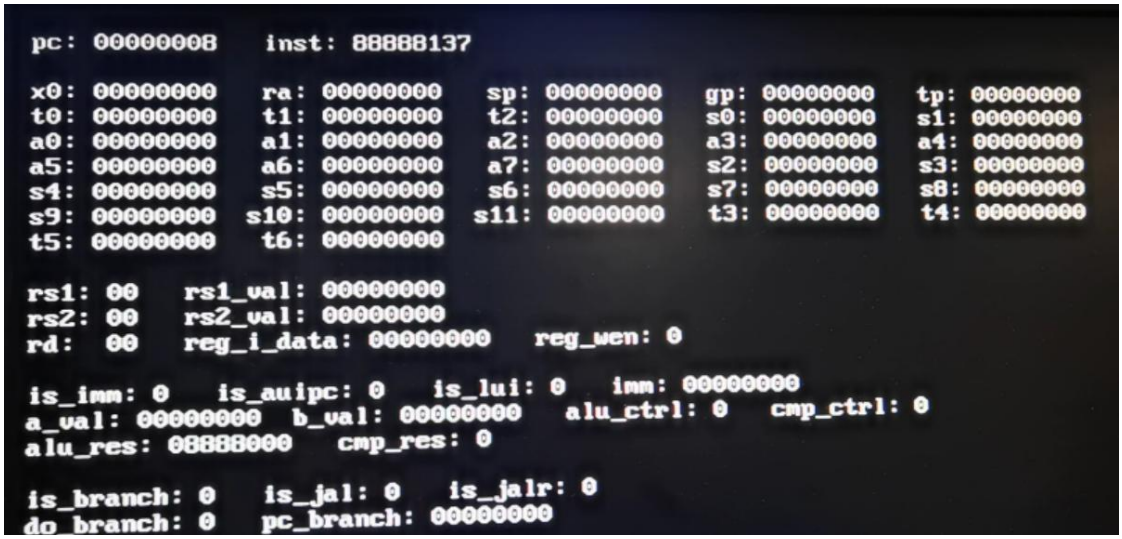

图 4.2.5 扩展指令测试结果（详见附件）

五、讨论与心得

1. 通过本次实验，我基本掌握了单周期 CPU 的运行流程和具体模块，对数据通路和控制通路的实现有了深入的理解，有助于课程知识的掌握。同时，通过几大模块的设计，更熟练掌握 RISC-V 各个类型的指令的功能，熟悉汇编指令的含义，能够更好地阅读、理解汇编指令，并能将汇编指令与机器码之间相互转换。

2. 本次实验中，在控制信号模块设计时，我遇到了较多困难，由于 PPT 存在一些笔误，对实验的推进造成了一些影响，通过仔细校对，最终解决了这些问题。而在扩展指令设计过程中，遇到了指令立即数错误的问题，最开始以为是立即数模块存在漏洞，但是最后发现原因在于之前修改完 SCPU 模块后 Vivado 并没有保存，导致信号宽度不一致，这一点上对后续实验有很大地警示作用。

3. 本次实验整体相对漫长，由于希望更好掌握，故完成了扩展指令与中断部分实验，以加深对知识的理解。