中国科学技术大学计算机科学与技术学院

《计算机组成原理》实验报告

实验题目：单周期 CPU 设计

学生姓名：郭耸霄

学生学号：PB20111712

完成日期：2022 年 3 月 22 日

计算机实验教学中心制

2020 年 09 月

实 验 报 告

11 系 20 级 3 班　　郭耸霄 PB20111712　　2022 年 3 月 22 日

1 实验题目

单周期 CPU 设计。

2 实验环境

开发板 Nexy4-DDR xc7a100tcsg324-1。

计算机 Windows Surface Pro 7 Model 1866 i7。

操作系统 Windows 11 Pro Version 22H2 (OS Build 22572.201)。

集成设计环境 Vivado 2019.1。

3 实验目的

1、理解 CPU 的结构和工作原理。
2、掌握单周期 CPU 的设计和调试方法。
3、熟练掌握数据通路和控制器的设计和描述方法。

4 实验步骤

设计单周期 CPU，并进行功能仿真 指令存储器和数据存储器：IP 例化的分布式存储器，容量均为 256x32 位，使用 Lab1 实验步骤 1 生成的 COE 文件初始化。

设计 PDU，并将 CPU 和 PDU 整合下载至 FPGA 中测试 寄存器堆和数据存储器各增加一个用于调试的读端口，使用 Lab1 实验步骤 2 生成的 COE 文件，对指令存储器和数据存储器初始化。

5 逻辑设计

5.1 数据通路

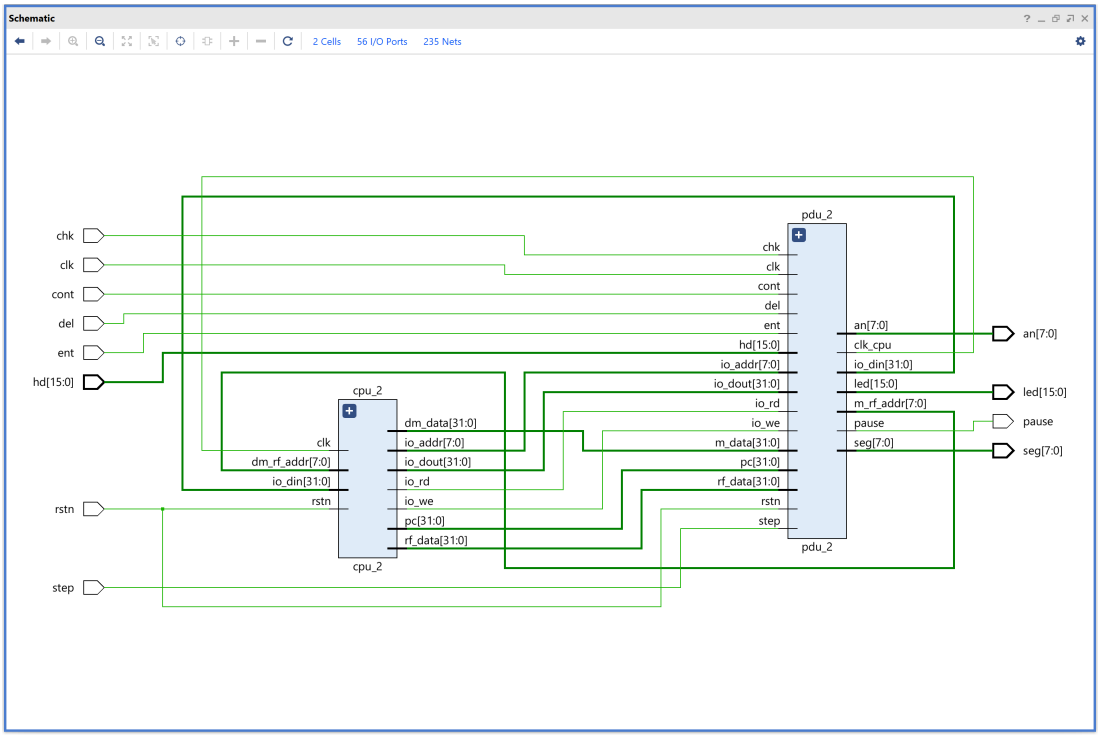

图 1: 顶层模块

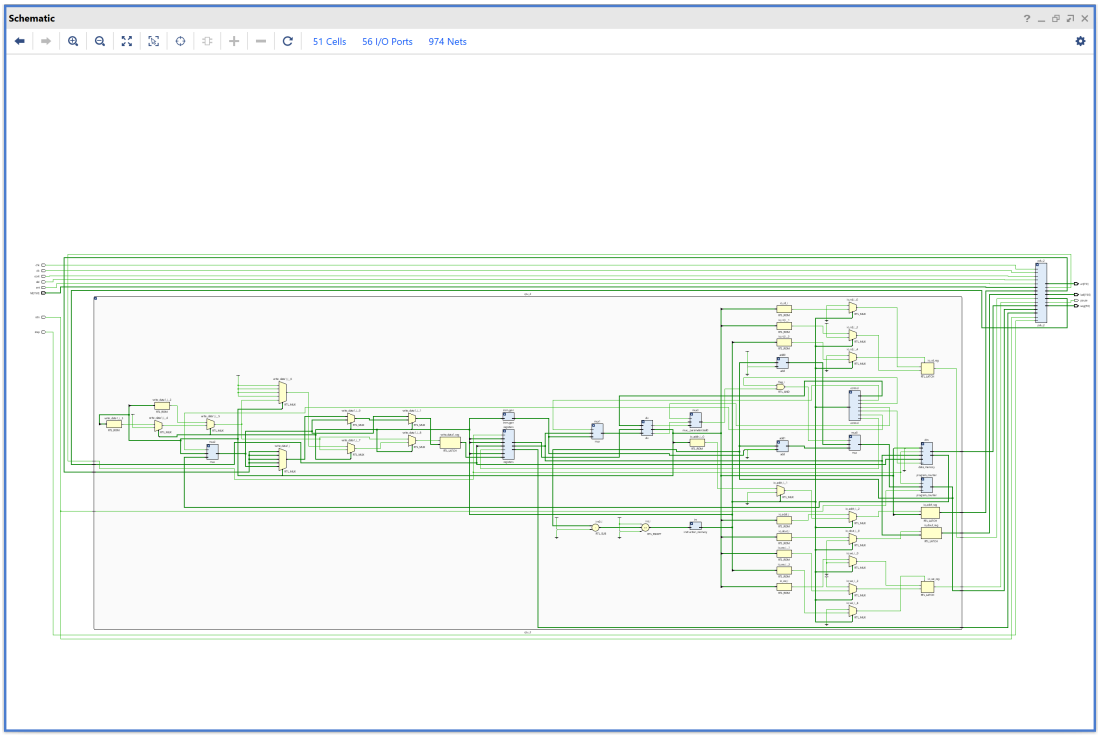

图 2: CPU 模块

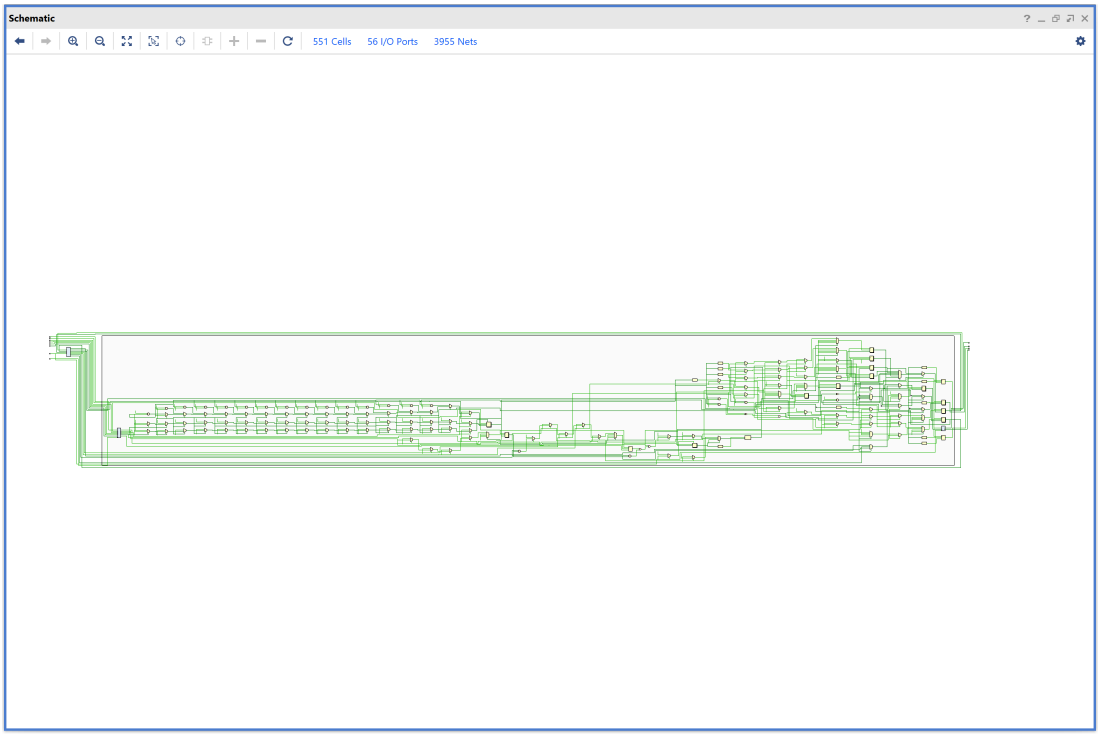

图 3: PDU 模块

5.2 状态机

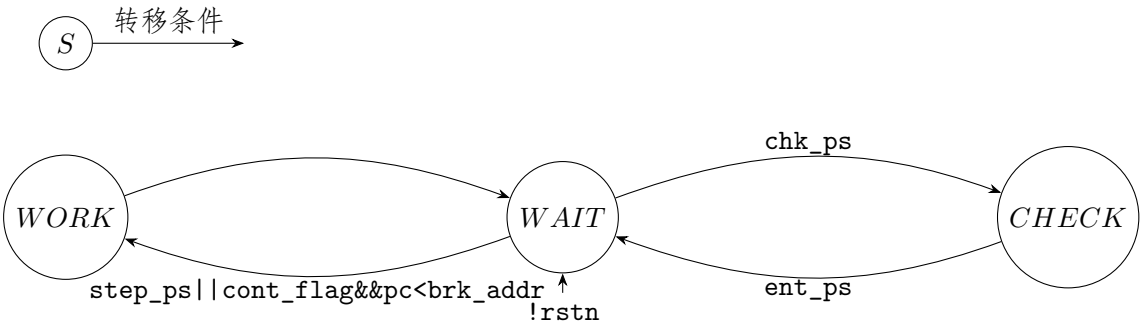

图 4: PDU 状态机

6 核心代码

这里仅放置 CPU 与 PDU 两个模块的代码，其余代码请看附件 0：实验源码。

CPU

```
1 `timescale 1ns / 1ps
2 //////////////////////////////////////////////////////////////////////////////////

3 // Company:
4 // Engineer:
5 //
```

```
6  // Create Date: 03/22/2022 08:44:56 AM
7  // Design Name:
8  // Module Name: cpu_2
9  // Project Name:
10 // Target Devices:
11 // Tool Versions:
12 // Description:
13 //
14 // Dependencies:
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 //////////////////////////////////////////////////////////////////////////////////


21
22
23 module  cpu_2 (
24   input  clk,
25   input  rstn,
26
27   //IO_BUS
28   output reg [7:0]  io_addr,  //led和seg的地址
29   output reg [31:0]  io_dout, //输出led和seg的数据
30   output  reg io_we,     //输出led和seg数据时的使能信号
31   input [31:0]  io_din, //来自sw的输入数据
32   output reg io_rd,    //输入数据时的读使能信号
33
34   //Debug_BUS
35   output [31:0]  pc,  //PC的内容
36   input [7:0]  dm_rf_addr,  //数据存储器DM或寄存器堆RF的调试读口
       地址
37   output [31:0]  rf_data, //从RF读取的数据
38   output [31:0]  dm_data  //从DM读取的数据
39 );
40 wire[31:0]c_pc,pc_plus4,n_pc,sum,alu_in1;
41 assign pc=c_pc;
42 program_counter program_counter(clk,rstn,n_pc,c_pc);
43 wire[31:0]instruction,imm_num,alu_result;
44 wire branch,mem_read,memto_reg,mem_write,alu_src,reg_write,flag,
     zero,cmp_func,branch_method,bigger;
45 wire [31:0]read_data0,read_data1,read_data2,write_data0,address;
```

```
46 wire[1:0]alu_op;
47 reg[31:0]write_data1;
48 registers registers(clk,reg_write,instruction[19:15],instruction
    [24:20],dm_rf_addr[4:0],instruction[11:7],write_data1,
    read_data0,read_data1,rf_data);
49 control control(instruction,branch,mem_read,memto_reg,mem_write,
    alu_src,reg_write,alu_op,cmp_func);
50 wire[7:0]ins;
51 assign ins=(c_pc-32'h3000)>>2;
52 instruction_memory im(.a(ins),.spo(instruction));
53 imm_gen imm_gen(instruction,imm_num);
54 add add0(c_pc,32'd4,pc_plus4);
55 add add1(c_pc,{imm_num[30:0],1'b0},sum);
56 and (flag,branch,branch_method);
57 mux mux0(pc_plus4,sum,flag,n_pc);
58 mux mux1(.in0(read_data1),.in1(imm_num),.out0(alu_in1),.flag(
    alu_src));
59 mux mux2(.in0(alu_result),.in1(read_data2),.out0(write_data0),.
    flag(memto_reg));
60 mux #(.W(1))mux3 (.in0(zero),.in1(bigger),.out0(branch_method),.
    flag(cmp_func));
61 initial begin
62   io_dout<=0;
63 end
64 always @(*) begin
65   if(instruction[6:0]===7'b0100011)begin
66     case(alu_result[7:0])
67     0:begin
68       io_addr<=0;
69       io_dout<=read_data1;
70       io_we<=1;
71     end
72     8:begin
73       io_addr<=8;
74       io_dout<=read_data1;
75       io_we<=1;
76     end
77     default:begin
78       write_data1=write_data0;
79     io_rd<=0;
80     io_we<=0;
81     end
82     endcase
```

```
83    end
84    else if(instruction[6:0]===7'b0000011)begin
85     case(alu_result[7:0])
86      4:begin
87        io_addr<=4;
88        write_data1<=io_din;
89        io_rd<=1;
90      end
91      12:begin
92        io_addr<=12;
93        write_data1<=io_din;
94        io_rd<=1;
95      end
96      16:begin
97        io_addr<=16;
98        write_data1<=io_din;
99        io_rd<=1;
100     end
101     20:begin
102       io_addr<=20;
103       write_data1<=io_din;
104       io_rd<=1;
105     end
106     default:begin
107      write_data1=write_data0;
108     io_rd<=0;
109     io_we<=0;
110     end
111     endcase
112   end
113   else begin
114     write_data1=write_data0;
115     io_rd<=0;
116     io_we<=0;
117   end
118 end
119 data_memory dm(.clk(clk),.a({2'b0,alu_result[7:2]}),.d(read_data1
      ),.dpra(dm_rf_addr),.dpo(dm_data),.spo(read_data2),.we(
      mem_write));
120 alu alu(read_data0,alu_in1,alu_op,alu_result,zero,bigger);
121 endmodule
```

PDU

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////
3  // Company:
4  // Engineer:
5  //
6  // Create Date: 03/22/2022 08:44:56 AM
7  // Design Name:
8  // Module Name: pdu_2
9  // Project Name:
10 // Target Devices:
11 // Tool Versions:
12 // Description:
13 //
14 // Dependencies:
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 //////////////////////////////////////////////////////////////////////////////////
21
22
23 module  pdu_2 (
24    input clk,      //clk100mhz
25    input rstn,     //cpu_resetn
26    input step,       //btnu
27    input cont,     //btnd
28    input chk,      //btnr
29    input ent,      //btnc
30    input del,      //btnl
31    input [15:0]  hd,     //sw15-0
32
33    output reg clk_cpu,
34    output reg pause,          //led16r
35    output reg[15:0] led,  //led15-0
36    output [7:0] an,      //an7-0
37    output [7:0] seg,     //ca-cg
38    //IO_BUS
39    input [7:0] io_addr,
40    input [31:0] io_dout,
41    input io_we,
```

```
42    output reg[31:0] io_din,
43    input io_rd,
44
45    //Debug_BUS
46    output reg[7:0] m_rf_addr,
47    input [31:0] rf_data,
48    input [31:0] m_data,
49    input [31:0] pc
50  );
51
52  parameter WAIT=0,CHK=1,OUT=2,IN=3,WORK=4;
53  reg[2:0]state,n_state;
54  reg[31:0]addr;
55  reg [31:0]brk_addr;
56  wire step_ps,cont_ps,chk_ps,ent_ps,del_ps;
57  wire[15:0]hd_ps;
58  db_ps db_ps(clk,step,cont,chk,ent,del,hd,step_ps,cont_ps,chk_ps,
      ent_ps,del_ps,hd_ps);
59  reg[31:0]show,temp;
60  dis dis(clk,rstn,show,an,seg);
61  integer i;
62  reg cont_flag,chk_flag;
63  reg[31:0]io_reg[5:0];
64  initial begin
65      addr=0;
66      cont_flag=0;
67      n_state=0;
68      state=0;
69      io_reg[0]=0;
70        io_reg[1]=0;
71         io_reg[2]=0;
72          io_reg[3]=0;
73           io_reg[4]=0;
74            io_reg[5]=0;
75  end
76  always @(posedge clk) begin
77      if(del_ps)begin
78              addr<={4'b0,addr[31:4]};
79          end else begin
80              for(i=0;i<16;i=i+1)begin
81                  if(hd_ps[i])begin
82                      addr<={addr[27:0],4'b0}+i;
83                  end
```

```
84              end
85          end
86          // if(del)begin
87          //      addr<={4'b0,addr[31:4]};
88          // end else begin
89          //      for(i=0;i<16;i=i+1)begin
90          //          if(hd[i])begin
91          //              addr<={addr[27:0],4'b0}+i;
92          //          end
93          //      end
94          // end
95      if(!rstn)
96          state<=WAIT;
97      else
98          state<=n_state;
99  end
100 always @(posedge clk) begin
101
102     case (n_state)
103     WAIT:begin
104         show<=io_reg[3]?io_reg[2]:addr;
105     clk_cpu<=0;
106      pause<=1;
107     led<=io_reg[0];
108     io_reg[1]<=hd_ps;
109     if(ent_ps)begin
110         // if(ent)begin
111         io_reg[3]<=0;
112         io_reg[5]<=0;
113     end else if(io_we)begin
114         if(io_addr==8)begin
115             io_reg[3]<=1;
116             io_reg[2]<=io_dout;
117
118         end else if(io_addr==0)
119             io_reg[0]<=io_dout;
120     end else if (io_rd)begin
121         if(io_addr==16)begin
122             io_reg[5]<=1;
123             io_reg[4]<=addr;
124             io_din<=io_reg[4];
125         end else if(io_addr==4||io_addr==12||io_addr==20)
126             io_din<=io_reg[io_addr[4:2]];
```

```
127     end else begin
128         if(chk_ps)
129             m_rf_addr<=addr[7:0];
130         else if(cont_ps)begin
131             cont_flag<=1;
132             brk_addr<=addr[15:0];
133         end else if(pc==brk_addr)begin
134            cont_flag<=0;
135         end
136         // if(chk)
137         //      m_rf_addr<=addr[7:0];
138         // else if(cont)begin
139         //      cont_flag<=1;
140         //      brk_addr<=addr[15:0];
141         // end   else if(pc==brk_addr)begin
142         //     cont_flag<=0;
143         // end
144     end
145     end
146     CHK:begin
147         led<=m_rf_addr;
148         if(step_ps)
149             m_rf_addr<=addr[7:0];
150         if(chk_ps)begin
151             case(addr[15:12])
152                 1:show<=pc;
153                 2:begin
154                     show<=rf_data;
155                     m_rf_addr<=m_rf_addr+1;
156                     end
157                 3:begin
158                     show<=m_data;
159                     m_rf_addr<=m_rf_addr+1;
160                 end
161                 default:;
162                 endcase
163         end
164     end
165     WORK:begin
166         clk_cpu<=1;
167         pause<=0;
168     end
169     endcase
```

```
170 end
171 always @(*) begin
172     case (state)
173     WAIT: begin
174         if(chk_ps)n_state=CHK;
175         else if(step_ps||cont_flag&&pc<brk_addr)n_state=WORK;
176         else n_state=WAIT;
177     end
178     CHK: begin
179         if(ent_ps)n_state=WAIT;
180         else n_state=CHK;
181     end
182     WORK: begin
183         n_state=WAIT;
184     end
185     endcase
186 end
187 endmodule
```

7 仿真结果

这是使用实验 1 中测试指令汇编程序对 CPU 运行仿真时屏幕截图。实际上，我还对整个 CPU-PDU 项目进行了许多次仿真。

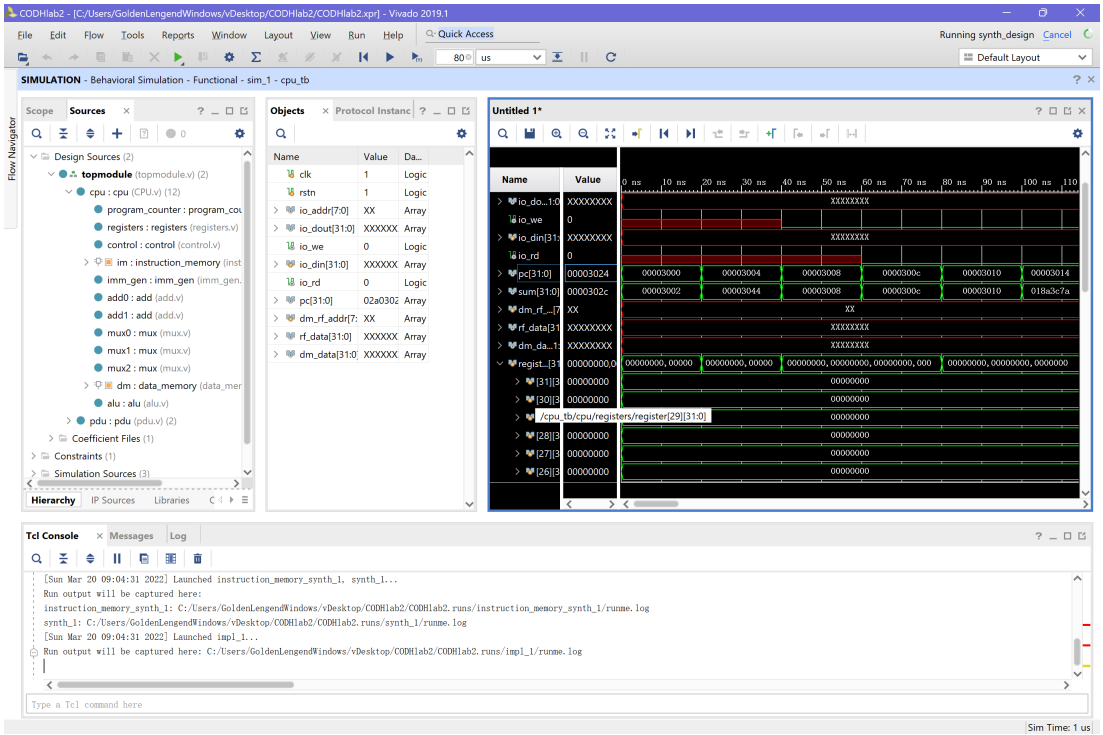

图 5: CPU 仿真结果

8 下载结果

请看附件 1：下载结果视频。

9 结果分析

结果符合设计预期。视频中原数据寄存器中存有的数组经过排序变成升序排列。进入检查模式，PC、寄存器堆、数据寄存器的数据均符合预期。

10 实验总结

10.1 实验过程

在实验中，我遇到了许多较难解决的错误，现列举如下：

现象 0 数码管动态显示只能显示 1 位。

问题 0 DIS 模块数码管选择信号位宽不匹配。

解决方案 0 在该数据位置用位拼接运算符占位。

现象 1 CPU 可以单步运行，但无法连续运行。

问题 1 判断运行终止终止采取了组合逻辑，导致直接运行停止。且传参数仅传 8 位，一定会小于 PC 首地址 3000。

解决方案 1 将 PDU 三段式输出部分改成时序逻辑，并将传参数改为 16 位。

现象 2 CPU 向下跳转时正常，向上跳转时异常。

问题 2 跳转时产生的立即数做了无符号拓展。

解决方案 2 在立即数产生器中使用有符号拓展产生立即数。

现象 3 输出时 io_dout 端口的值突变为异常值。

问题 3 CPU 的输出部分采用组合逻辑判断，导致毛刺的产生。以及数码管多用途显示的逻辑有错误。

解决方案 3 采用时序逻辑选择 CPU 的输出数据，并且重新设计了数码管多用途显示的逻辑。

现象 4 汇编程序没有进入任何循环。

问题 4 寄存器及存储器读取单元时有时以 1 为单位，有时以 4 为单位。本质是汇编程序设计的存储单位是 1 字节，但是 CPU 的存取单位是 4 字节。

解决方案 4 将地址寄存器的输出统一右移两位。

10.2 实验检查

2022-03-21 日，我本以为已经完成了实验要求，便前往电三楼找樊老师检查实验。途中看到张老师，便上前询问我不理解的轮巡输出部分。我开始不理解的是 CPU 的 IO 总线只有一个数据输入，该如何判断 PDU 与人交互时的读写完毕标志呢？经过张老师的讲解，我明白了原来读写完毕标志也是与输入输出数据同样的数据，也可以通过 io_din 读取。而我实现的属于中断输出，与实验要求有所出入。于是我又修改了设计文件，得到了现在的结果。但是我没有丢弃我的中断输入输出模块，希望可以将它作为本次实验的拓展部分。

10.3 实验收获

通过这次实验，我独立地运用了之前许多数字电路及计算系统课程学习的知识，完成了一个小型的 CPU 的设计，可以说终于解决了我持续 2 年的关于 “程序是如何运行的” 这一问题，并且对计算机组成原理有了更进一步的认识，也对后续 CPU 的设计实验有了更强的信心。

11 意见建议

时间安排 建议本实验安排两周。或者将第一次与第二次实验合为一次，用两周完成。

内容安排 本次实验制作的是哈佛架构构的 CPU，若运行 lab1 的在冯 · 诺依曼架构的汇编程序，需要重新编辑。并且要求的 6 条指令几乎无法完成排序功能，虽然可以自己添加指令，但是已给出的数据通路会对此阶段造成一定的限制，应该要求实现 bequ 这条指令，并给出完整的数据通路，或提示该数据通路仅作参考。