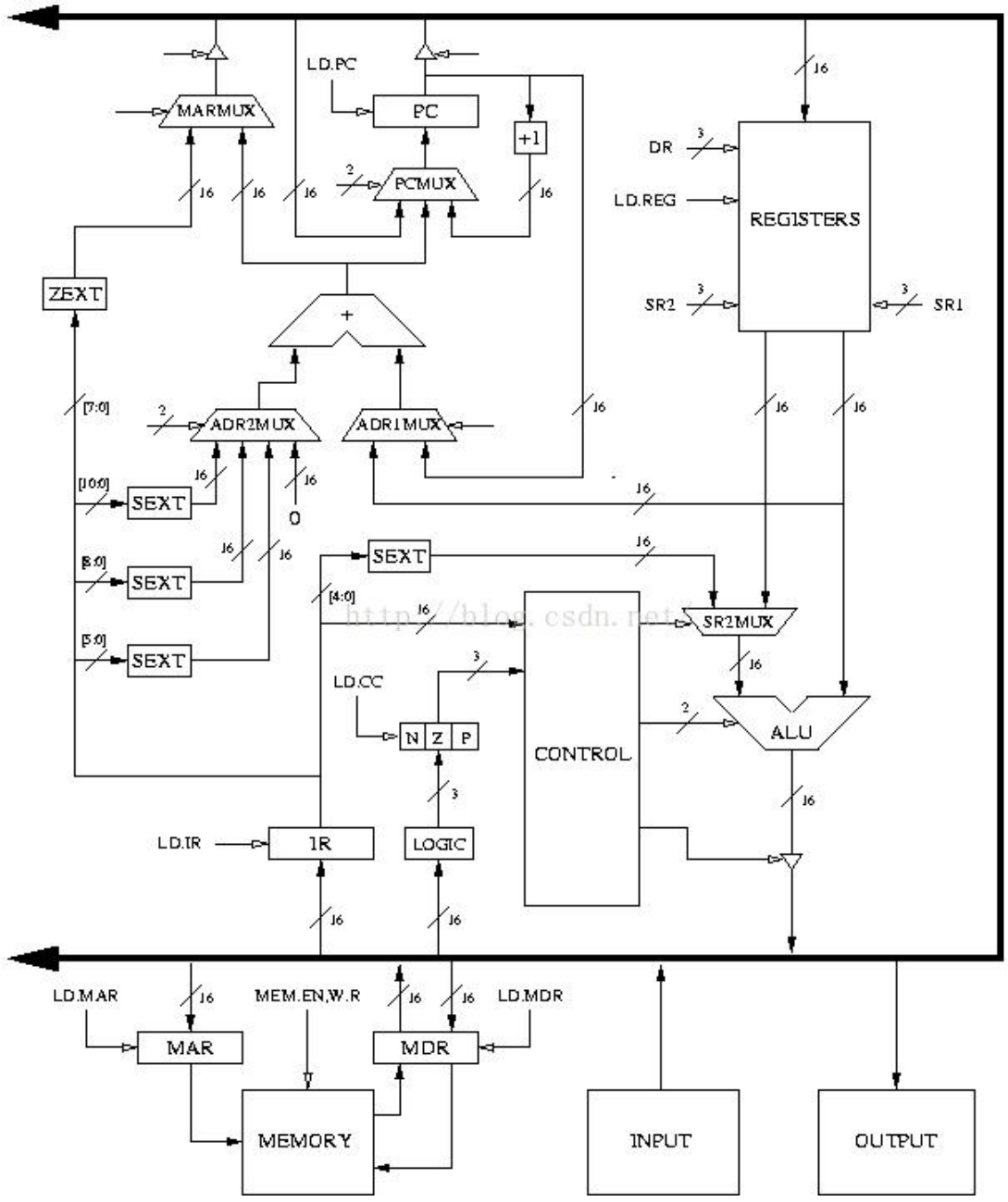
Lab11 设计说明书

PB17050941 李喆昊

一．电路功能：《计算机系统概论》LC-3 FPGA 实现

1. 整体架构：基于下图 LC-3 Data Path



图一 LC-3 Data Path

（1） 设计与实验平台：Vivado 2018 , Nexys4 DDR

编程语言: verilog

（2）输入

使用开关（swt[15:0]）输入指令、BTNC 作为状态 reset 按钮、BTNL 作为 execute_clk 信号。

（3）输出

使用七段数码管、led[15:0]、两个 RGB led 灯。

（4）寄存器与内存

① 寄存器文件（Register File）与《计算机系统概论》课本内容一致，为[15:0]R[7:0]

② 条件码寄存器 n，z，p

③ 内存： 256 x 1byte（地址：x0000--x00FF）

④ 进行控制的重要寄存器：

IR : Instruction Pointer , 用于存放指令

PC：Program Counter，指向内存中待执行的下一条指令

MDR : Memory Data Register，存放内存与外部进行交换的数据

MAR : Memory Data Register，存放需要被访问的内存单元的地址

（5）控制

① 两种控制模式：

根据开关输入内容的不同，可以选择两种控制模式：

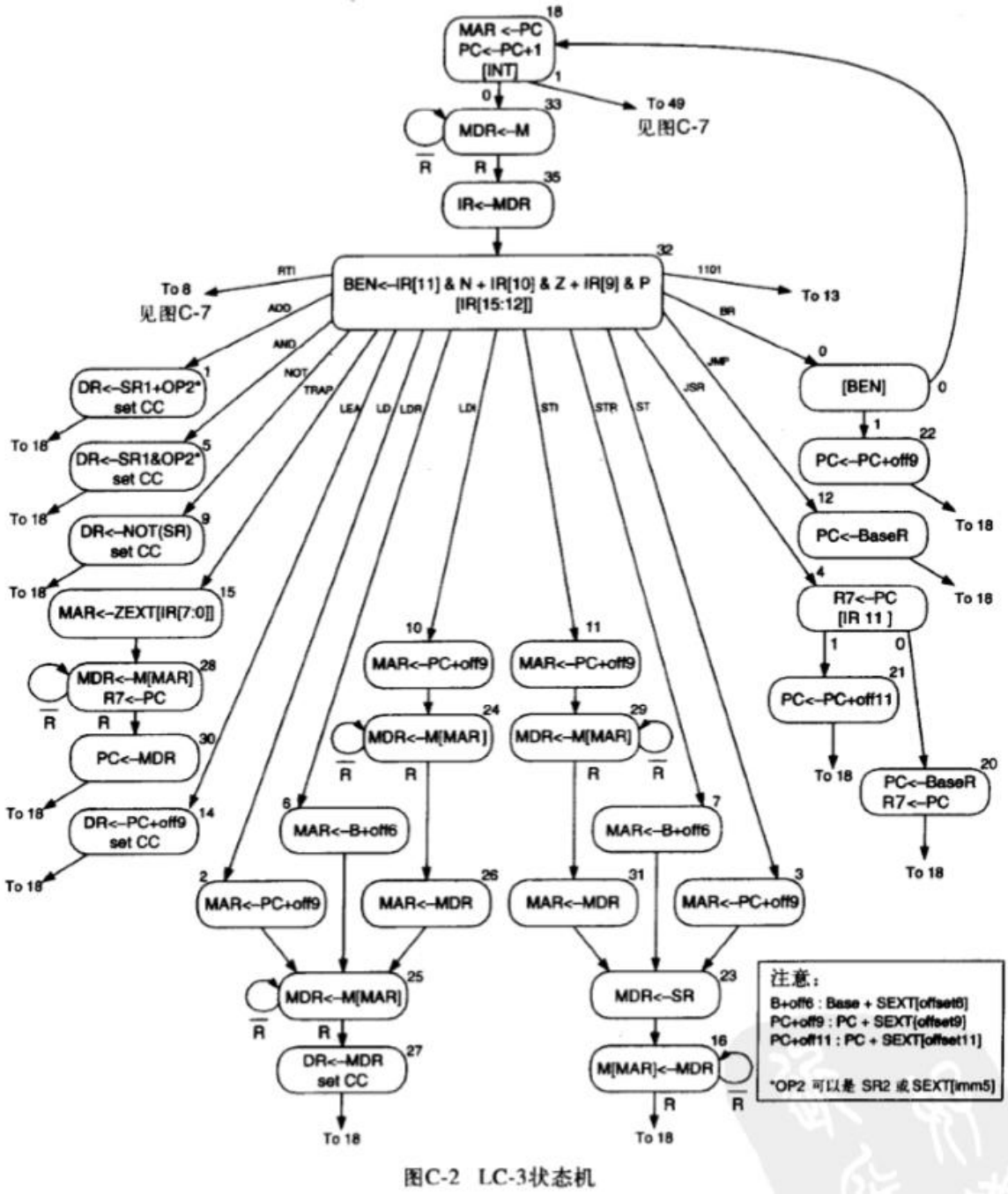
A. 指令直接由开关输入 IR 并执行：

当 led 灯只有 led0 亮起时（状态 S1），依据要执行的指令拨动开关 swt[15:0]，再按下 BTNL 按钮，指令直接输入 IR，接下来译码（状态 S3）并执行。

B. 使用 PC 从内存中读取指令执行：

在内存中预先存好要执行的程序。当 led 灯只有 led0 亮起时（状态 S1），拨动开关至 1101_111x_xxxx_xxxx，再按下 BTNL 按钮，进入 RUN 模式（状态 S0），即使用 PC 从内存中逐条读取指令执行。此时七段数码管显示“RU”和当前 PC 的值，同时两个 RGB led 灯亮黄色。（图见后 3（1）E 部分）

② 具体控制使用 3 段式有限状态机，基于 LC-3 有限状态机编写。



图C-2 LC-3状态机

图 2 LC-3 状态机

2. LC-3 指令集的实现

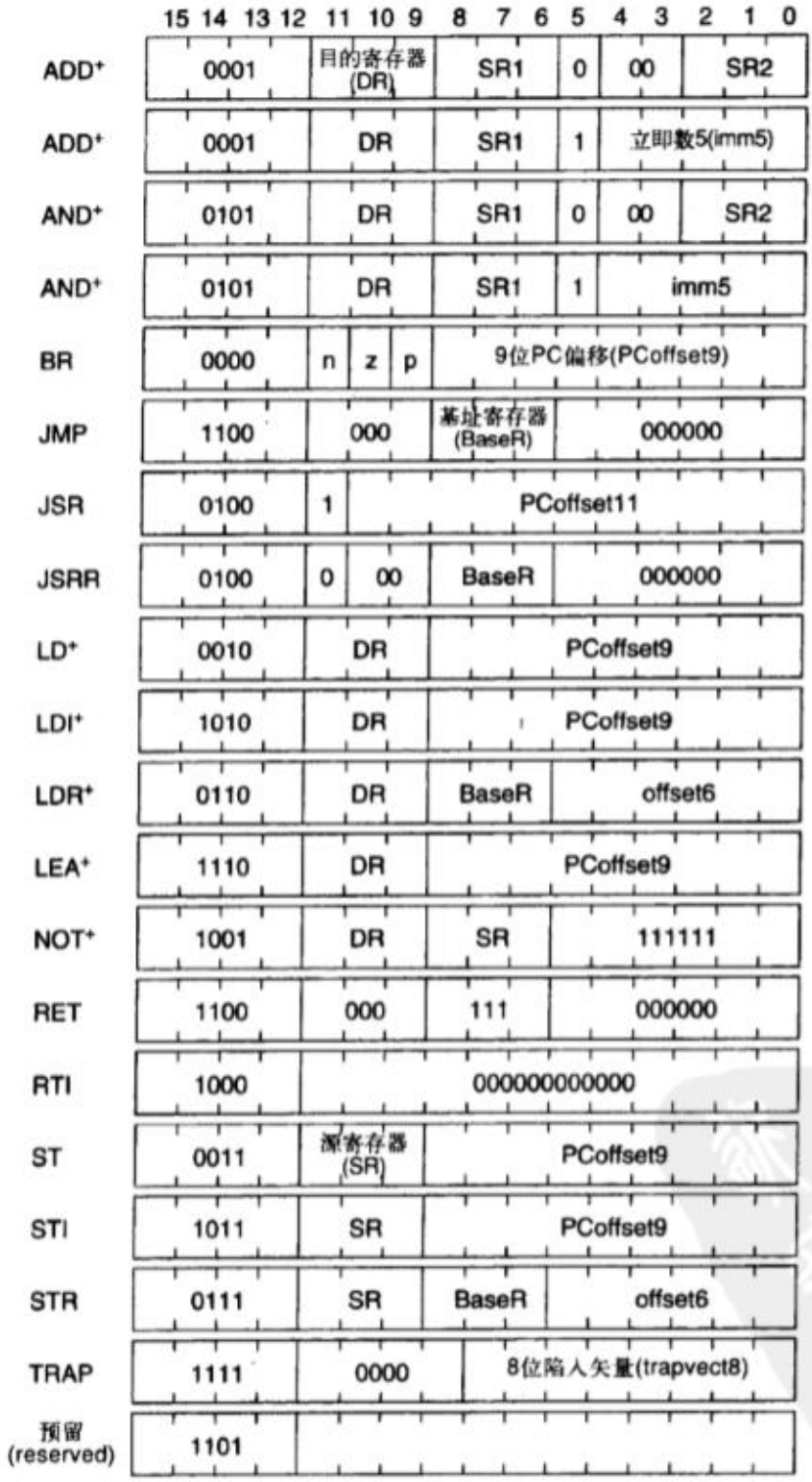


图5-3 LC-3指令集（全部）的格式。注意，+表示该指令将改变条件码

图 3　LC-3 ISA

（1）预留的 1101 指令

我使用其作为调用交互界面及切换控制模式的指令。

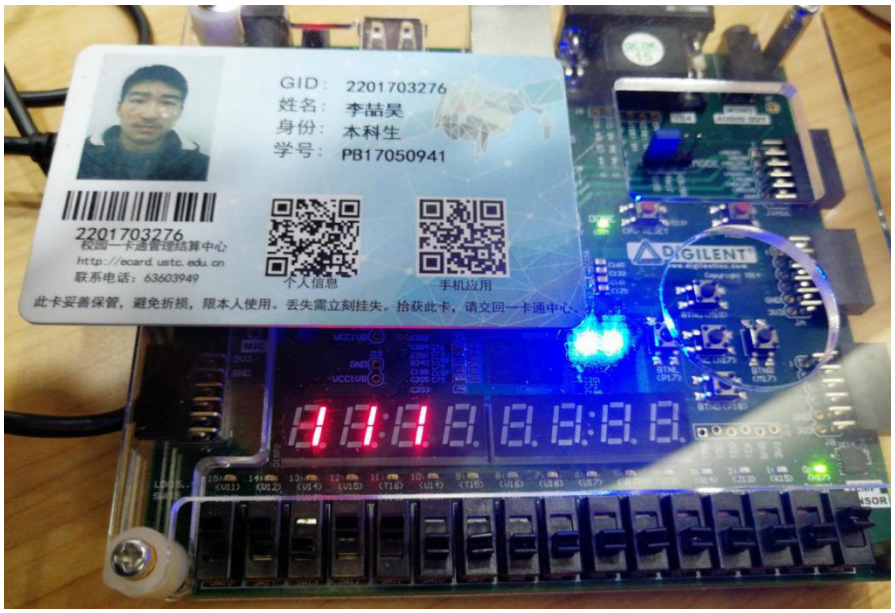
A. 1101_0xxx_xxxxx_RegNum[3:0]　输出寄存器 R0--R7 的内容

上图：在七段数码管上显示寄存器 R7 的值：xFFFF。同时两个 RGB led 灯亮绿色，表示当前是查看寄存器模式

B. 1101_110_xxxxx_xxxx　输出内存中的内容

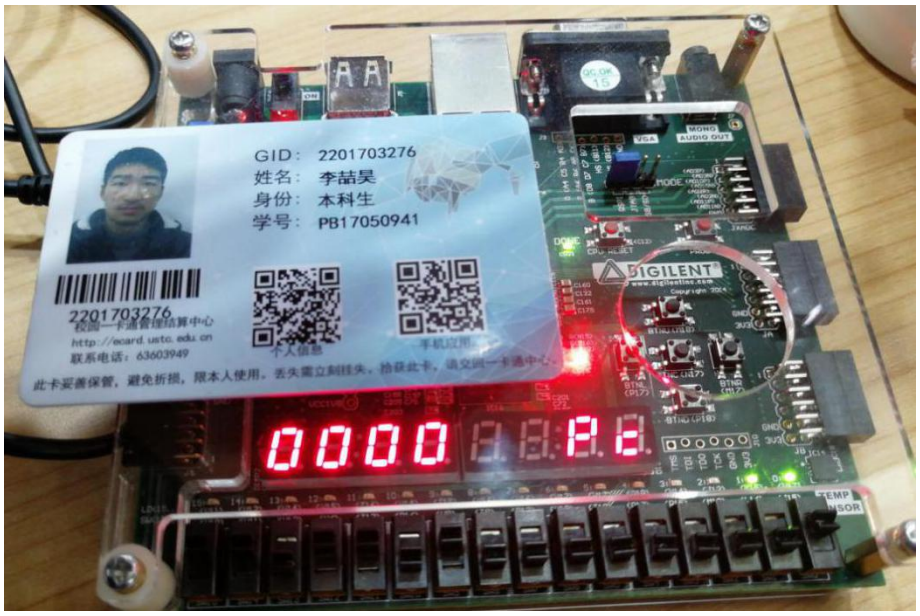
上图：七段数码管上显示 memory[x0001]内存单元的值：x1422（预先存入）。同时两个 RGB led 灯亮蓝色，表示当前是查看内存模式

C. 1101_100x_xxxx_xxxx: 输出条件码寄存器 nzp 的值



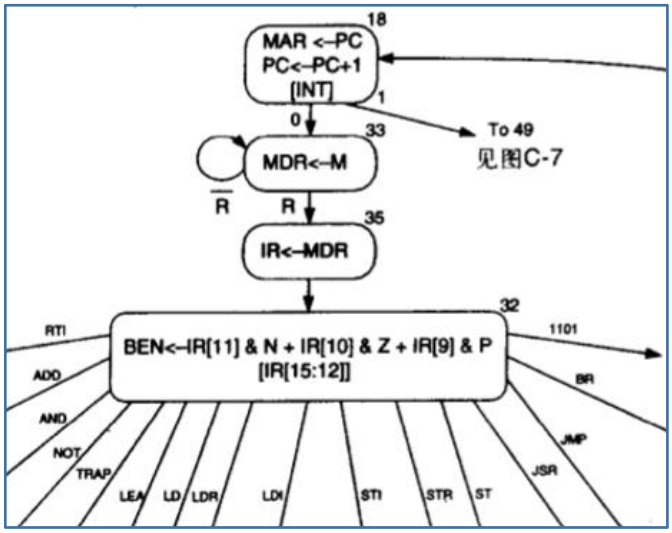
上图：七段数码管上显示条件码寄存器 n，z，p 的值：1，1，1

D. 1101_101x_xxxx_xxxx: 输出 PC 的值

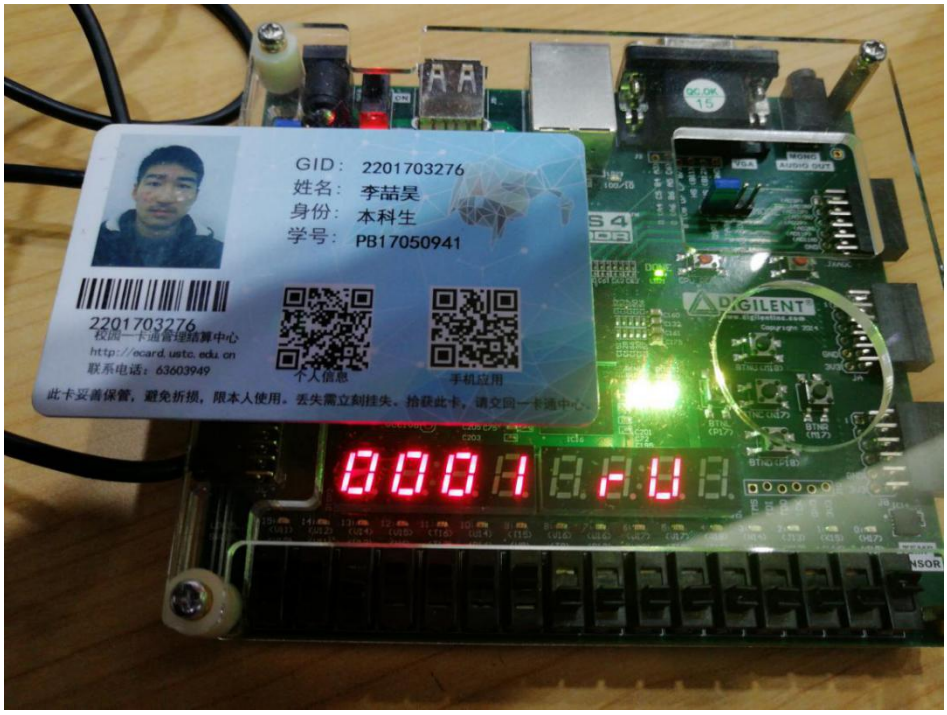


上图：七段数码管上显示当前 PC 的值：x0000。同时两个 RGB led 灯亮红色，表示当前是查看 PC 模式

E. 1101_111x_xxxx_xxxx: 切换控制模式为从内存中读取指令模式



过程如上图，使用 PC、MAR、MDR 从内存中读取指令存入 IR，然后译码执行

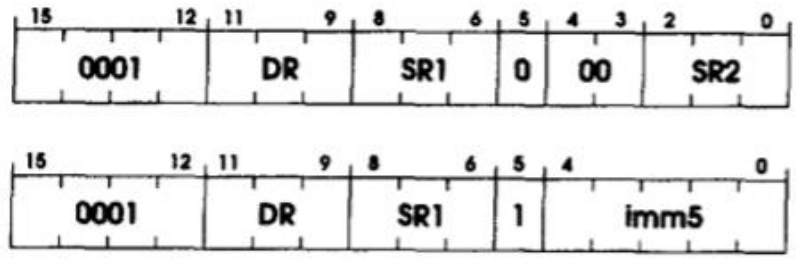


上图：七段数码管显示 RU 同时两个 RGB led 灯亮黄色，表示当前为

内存读取指令模式（RUN 模式），PC=PC+1（由 x0000 变为 x0001）

（2）ADD 指令

编码

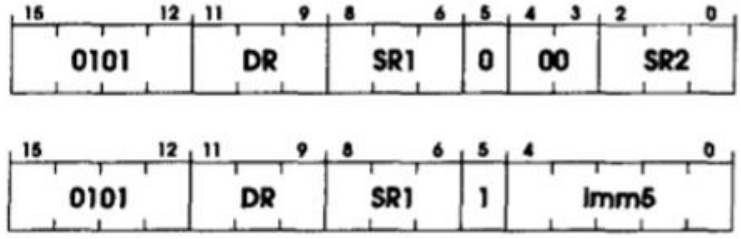


指令的第五位为 1 表示第二个操作数为立即数，否则为寄存器

初始，R1=x00C5，令 R1=R7+0，将 R7 的值（xFFFF）转移至 R1 中

（3）AND 指令

编码

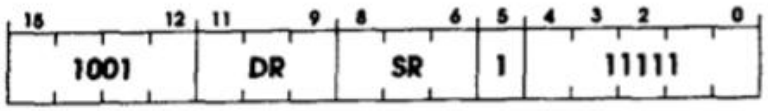


与 ADD 同，指令的第五位为 1 表示第二个操作数为立即数，否则为寄存器

将 R1 与 x000F 作与运算，R1 变为 x000F

（4）NOT 指令

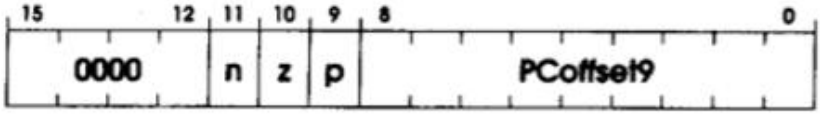
编码



将非 R1 结果（x000F--->xFFF0）存入 R2 中，如图

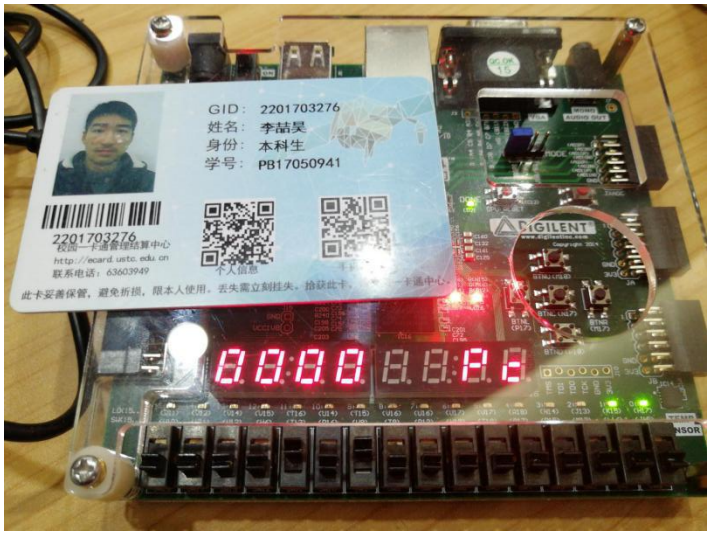
（5）BR 指令

编码



BR 指令是一种重要的分支跳转指令。根据条件码寄存器的不同来改变 PC 的值：PC=PC+offset

条件码为 001，PC 初始为 x0000（下图）



输入 BR 指令：0000_001_0000_00011, PC=PC+3，下图中 PC 变为 3

（6）JMP 指令

编码

	15 12	11 9	8 6	5 0
JMP	1100	000	BaseR	000000

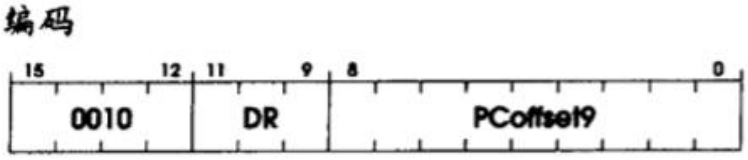
JMP 是一种重要的无条件跳转指令

R5=x000F，使用 JMP 指令，PC=R5=x000F

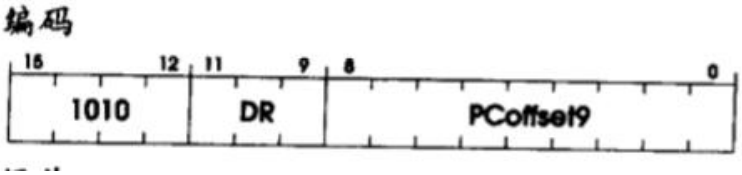
因为 RET 指令实际上就是 JMP R7，所以 RET 指令也已经实现了。

（7）LD，LDI，LDR 指令

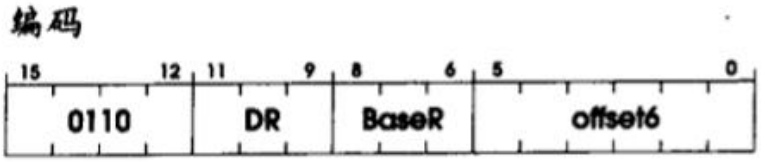
LD： 将 memory[PC+offset]传给 DR 寄存器



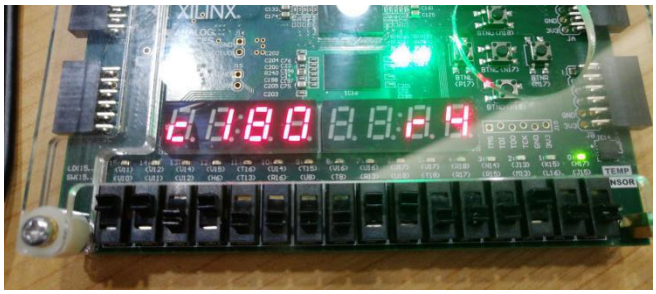
LDI：将 memory[PC+offset]的内容作为地址，将该地址的 memory 内容传给 DR 寄存器



LDR：将 memory[BaseR+offset]传给 DR 寄存器

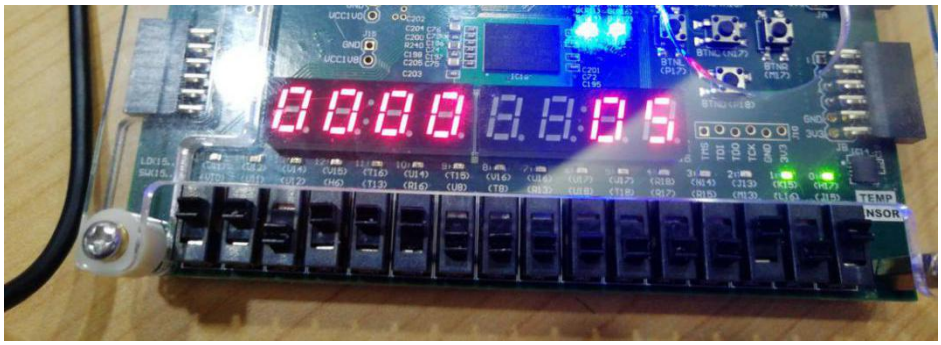


Memory[x0002]=xC180，R4 初始为 x0000，R3 为 x0002，使用 LDR 指令将内存内容 xC180 传到 R4



（8）ST，STI，STR 指令

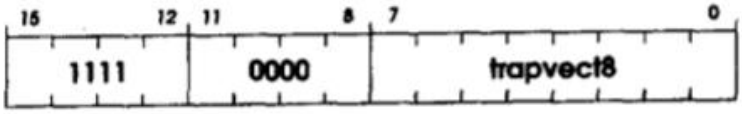
这三条 store 指令与 load 指令是对称的。



初始 memory[x0005]=x0000,使用 ST 指令将该内存地址储存了 R4 的值：xC180.

（9）TRAP 指令

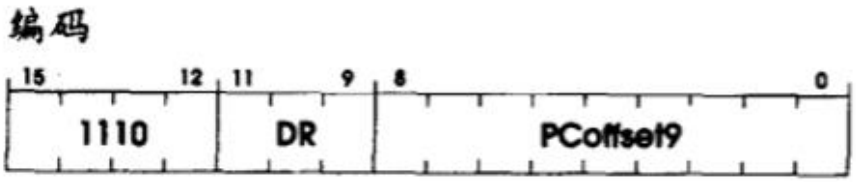
编码



由于涉及到操作系统服务例程，较复杂，故只实现了 HALT 功能：

当前执行的指令为 1111 指令时，led 等全亮。表示 HALT，程序执行完毕。

（10）LEA 指令



LEA 可用于将一个内存地址存入 DR 寄存器：DR=PC+PCoffset

R6 初始为 x0005，PC=x0002，使用 LEA 指令，将 R6 变为 PC+7=x0009

（11）JSR，JSRR，RTI 指令

因为这些指令涉及到调用子程序以及中断的问题，较复杂，故没有实现。

总体上看，LC-3 大部分指令都已实现，基本完成实验目标。

二．源代码分析

1. 三段式有限状态机

```
//第一个always： 状态转换always
always @(posedge execute_clock or posedge state_reset)begin
      if(state_reset) state=S1;
      else state=nextstate;
end
```

第一个 always：在 execute_clock 的每个上升沿，状态发生转换。

```
//第二个always： 状态内容
always @(state) begin
      case(state)
        //****************  S0 从内存中取指令 ****************
        S0: begin
           led=S0;
           IR=memory[PC];
           PC_record=PC+16'b1;
           nextstate=S6;
        end
        //*************** S1 swt直接输入IR
        S1:begin
           led=S1;   //led用于指示工作状态
           IR=swt;
           nextstate=S3;
        end
```

```
//********************** S3: 译码器 **********************
S3: begin
   led=S3;
   case(IR[15:12])  //前4位为opcode
    // ####1. OUTPUT####  1101_0xxx_xxxxx_RegisterNum[2:0]
    //                or  1101_110x_MemoryAddress[7:0],
    //                or 1101_100x_xxxx_xxxx(输出nzp), 1101_101x_xxxx_xxxx(输出PC)
    //                or 1101_111_xxxxxxxx ;执行PC
        4'b1101: begin
           if(IR[11]==0) begin  //输出寄存器
                num_reg_show={1'b0,IR[2:0]};
                output_memory=1'b0;
                output_MP=1'b0;
                output_PC=1'b0;
                output_run=1'b0;
                nextstate=S1;
           end
           else begin
               case(IR[11:9])
                       3'b110: begin  //输出内存内容
                                address_memory_show=IR[7:0];  //8位地址
                                output_memory=1'b1;
                                output_MP=1'b0;
                                output_PC=1'b0;
                                output_run=1'b0;
                                nextstate=S1;
                        end
```

```
                3'b100: begin   //输出nzp
                        output_memory=1'b0;
                        output_MP=1'b1;
                        output_PC=1'b0;
                        output_run=1'b0;
                        nextstate=S1;
                end
                3'b101:begin   //输出PC
                        output_memory=1'b0;
                         output_MP=1'b0;
                         output_PC=1'b1;
                         output_run=1'b0;
                         nextstate=S1;
                end
                3'b111: begin
                        output_memory=1'b0;
                        output_MP=1'b0;
                        output_PC=1'b0;
                        output_run=1'b1;
                        nextstate=S0;
                end
                default:;
            endcase
    end
end
```

状态 S0 与 S1 均用于设置 IR，但两者对应的控制模式不同：S0 对应从内存读取指令模式，S1 对应指令由开关直接输入 IR 模式。

S3 是译码器，根据 IR[15:12]（opcode）的内容决定下一个状态是什么以及要执行什么操作。

以下均为 S3 的内容，显示了对 ADD、AND、BR 等指令的译码和执行过程。

```
//##### 2.ADD ###### 0001_DR_SR1_xxx_SR2 or 0101_DR_SR1_1_imm5
4'b0001: begin
       case(IR[5])
           1'b1:  //立即数模式
           begin
                 SEXT={{11{IR[4]}},IR[4:0]};
                 add_temp=( R[IR[8:6]] + SEXT);
           end
           //寄存器模式
           default: begin
               add_temp=(R[IR[8:6]]+R[IR[2:0]]);
           end
       endcase
       R[IR[11:9]]=add_temp;
       //设置条件寄存器
       if(R[IR[11:9]]>16'h0000) {n,z,p}={1'b0,1'b0,1'b1};
       else if (R[IR[11:9]]==16'h0000) {n,z,p}={1'b0, 1'b1,1'b0};
       else {n,z,p}={1'b1,1'b0,1'b0};

       nextstate=S1;
   end

//##### 3.AND ###### 0101_DR_SR1_xxx_SR2 or 0101_DR_SR1_1_imm5
4'b0101:begin
    case(IR[5])
       //立即数模式
       1'b1: R[IR[11:9]] = (R[IR[8:6]] &{ {11{IR[4]}} ,IR[4:0]});  //用{{}}进行SEXT
       //寄存器模式
       default: R[IR[11:9]]=(R[IR[8:6]] & R[IR[2:0]]);
   endcase
   //设置条件寄存器
   if(R[IR[11:9]]>16'h0000) {n,z,p}={1'b0,1'b0,1'b1};
   else if (R[IR[11:9]]==16'h0000) {n,z,p}={1'b0, 1'b1,1'b0};
   else {n,z,p}={1'b1,1'b0,1'b0};

   nextstate=S1;
end
```

```
// #####4.NOT #### 1001_DR_SR_xxxxxx
4'b1001: begin
   not_temp=(~ R[IR[8:6]]);
   R[IR[11:9]]=not_temp;
   nextstate=S1;

    //设置条件寄存器
   if(R[IR[11:9]]>16'h0000) {n,z,p}={1'b0,1'b0,1'b1};
   else if (R[IR[11:9]]==16'h0000) {n,z,p}={1'b0, 1'b1,1'b0};
   else {n,z,p}={1'b1,1'b0,1'b0};

end


    // ###### 5.BR ##### 0000_nzp_PCOffset[8:0]
    4'b0000: begin
        if({n&IR[11],z&IR[10],p&IR[9]})  begin
            add_temp=PC+{{7{IR[8]}},IR[8:0]};
            nextstate=S4;
        end
        else begin
            PC=PC;
            nextstate=S1;
        end
    end


    //###### 6.JMP  1100_xxx_BaseR[3:0]_xxxxxx
    4'b1100:begin
        PC=R[IR[8:6]];
        nextstate=S1;
    end
    //##### 7.LD  0010_DR[3:0]_PCOffset9
    4'b0010: begin
        R[IR[11:9]]=memory[PC+{{7{IR[8]}},IR[8:0]}];

        if(R[IR[11:9]]>16'h0000) {n,z,p}={1'b0,1'b0,1'b1};
        else if (R[IR[11:9]]==16'h0000) {n,z,p}={1'b0, 1'b1,1'b0};
        else {n,z,p}={1'b1,1'b0,1'b0};

        nextstate=S1;
    end
```

```
//###### 8.LDR 0110_DR_BaseR_offset6
4'b0110: begin
    R[IR[11:9]]=memory[R[IR[8:6]]+{{10{IR[5]}},IR[5:0]}];

    if(R[IR[11:9]]>16'h0000) {n,z,p}={1'b0,1'b0,1'b1};
    else if (R[IR[11:9]]==16'h0000) {n,z,p}={1'b0, 1'b1,1'b0};
    else {n,z,p}={1'b1,1'b0,1'b0};
    nextstate=S1;
end

//###### 9 LDI 1010_DR_PCoffset9;
 4'b1010: begin
    temp_address=memory[PC+{{7{IR[8]}},IR[8:0]}];
    R[IR[11:9]]=memory[temp_address];

    if(R[IR[11:9]]>16'h0000) {n,z,p}={1'b0,1'b0,1'b1};
    else if (R[IR[11:9]]==16'h0000) {n,z,p}={1'b0, 1'b1,1'b0};
    else {n,z,p}={1'b1,1'b0,1'b0};
    nextstate=S1;
end

//###### 10.LEA 1110_DR_PCoffset9
4'b1110:begin
    R[IR[11:9]]=PC+{{7{IR[8]}},IR[8:0]};

     if(R[IR[11:9]]>16'h0000) {n,z,p}={1'b0,1'b0,1'b1};
     else if (R[IR[11:9]]==16'h0000) {n,z,p}={1'b0, 1'b1,1'b0};
     else {n,z,p}={1'b1,1'b0,1'b0};
     nextstate=S1;
end
//####### 11.ST 0011_SR_PCOffset9
4'b0011: begin
       memory[PC+{{7{IR[8]}},IR[8:0]}]=R[IR[11:9]];
       nextstate=S1;
end
//######  12 STR 0111_SR_BaseR_offset6;
4'b0111:begin memory[R[IR[8:6]]+{{10{IR[5]}},IR[5:0]}] = R[IR[11:9]];
        nextstate=S1;
end
```

```
//######  13.STI 1011_SR_PCOffset9
4'b1011:  begin
       temp_address=memory[PC+{{7{IR[8]}},IR[8:0]}];
       memory[temp_address]=R[IR[11:9]];
       nextstate=S1;
end
//##### 14 HALT
4'b1111: begin
    led=16'hffff;
    nextstate=S1;
end
default: begin
    num_reg_show=4'b0000;
    nextstate=S1;
end
```

S4，S5 是控制 PC 自增与加 offset 的辅助状态：

```
            //S4
            S4: begin  //PC
                   led=S4;
                   PC=add_temp;
                   nextstate=S1;
            end
            S6:begin  //S0
               led=S6;
               PC=PC_record;
               //if(PC>PC_record) PC=PC_record;
               nextstate=S3;
            end
            default:nextstate=S1;
           endcase

    end
```

第二个 always 到这里结束。

第三个 always 用于控制输出，见下面输出部分。

2. 输出部分

（1）七段数码管输出模块

```
module segment_decoder(
   input [4:0]y,
   output reg [6:0]x
   );
   always @ *
       case(y[4])
           0:begin
               case(y[3:0])
                   4'b0000: x = 7'b100_0000; //0000_001;   //case can only be used to reg!!
                   4'b0001: x = 7'b111_1001;//1001_111;
                   4'b0010: x = 7'b010_0100;//0010_010;
                   4'b0011: x = 7'b011_0000;//0000_110;//
                   4'b0100: x = 7'b001_1001;//1001_100;
                   4'b0101: x = 7'b001_0010;//0100_100;
                   4'b0110: x = 7'b000_0010;//0100_000;
                   4'b0111: x = 7'b111_1000;//0001_111;
                   4'b1000: x = 7'b000_0000;//0000_000;
                   4'b1001: x = 7'b001_0000;//0000_100;
                   4'b1010: x= 7'b000_1000; //A 000_1000
                   4'b1011: x= 7'b000_0011; //b 110_0000
                   4'b1100: x= 7'b010_0111; //c 111_0010
                   4'b1101: x= 7'b010_0001; //d 100_0010
                   4'b1110: x= 7'b000_0110;//E  011_0000
                    default: x= 7'b000_1110;//F 011_1000
               endcase
           end
             1: begin
                 case(y[3:0])
                      4'b0000:x=7'b010_1111;//r  111_1010
                      4'b0001:x=7'b000_1100;// P
                      4'b0010: x=7'b100_0001; // u
                      4'b0011: x=7'b100_1000;  // n
                      4'b0100: x=7'b100_1111;// I
                      default:;
                 endcase
            end
             endcase
endmodule
```

（2）输出信号，根据指令 1101 选择显示的内容（查看寄存器、内存、PC、nzp、RUN 模式）

使用了 FPGA 自带的 5GHz 时钟，并使用 cnt_show 变量分频

```
always @(posedge Clk5)begin
    if(cnt_show >= 13'd5999)
        cnt_show    <= 13'h0;
    else
        cnt_show    <= cnt_show + 13'h1;
    case(output_memory)
        0: begin
                if(output_MP==1'b1) begin  //输出nzp内容

                    if(cnt_show<=13'd999)begin
                            an= 8'b0111_1111;  show=n;//show=IR[15:12];
                        end
                    else if(cnt_show<=13'd1999)begin
                            an= 8'b1011_1111;  show=z;//show=IR[11:8];
                        end
                    else if(cnt_show<=13'd2999)begin
                            an= 8'b1101_1111; show=p;//show=IR[7:4];
                        end
                    else if(cnt_show<=13'd3999)begin
                            an= 8'b1111_1111; //show=IR[3:0];
                        end
                    else if(cnt_show<=13'd4999)begin
                            an= 8'b1111_1111; //show=5'b10100; //I
                        end
                    else begin
                            an= 8'b1111_1111; //show= 5'b10000; //r
                        end
                end
```

上图是在七段数码管上显示 nzp 的部分控制代码

```
else if(output_PC==1'b1)begin  //输出PC内容
        led16=3'b100;
        led17=3'b100;
        if(cnt_show<=13'd999)begin
                    an= 8'b0111_1111;  show=PC[15:12];
                end
            else if(cnt_show<=13'd1999)begin
                    an= 8'b1011_1111;  show=PC[11:7];
                end
            else if(cnt_show<=13'd2999)begin
                    an= 8'b1101_1111; show=PC[7:4];
                end
            else if(cnt_show<=13'd3999)begin
                    an= 8'b1110_1111; show=PC[3:0];
                end
            else if(cnt_show<=13'd4999)begin
                    an= 8'b1111_1101; show=5'b10001;  //r
                end
            else begin
                    an= 8'b1111_1110; show=5'b01100; //C
                end
end
```

上下图分别是在七段数码管上显示 PC、RUN 模式的部分控制代码

```
else if(output_run==1'b1)begin  //输出RUN
        led16=3'b110;
        led17=3'b110;
        if(cnt_show<=13'd999)begin
                    an= 8'b0111_1111;  show=PC[15:12];
                end
            else if(cnt_show<=13'd1999)begin
                    an= 8'b1011_1111;  show=PC[11:7];
                end
            else if(cnt_show<=13'd2999)begin
                    an= 8'b1101_1111; show=PC[7:4];
                end
            else if(cnt_show<=13'd3999)begin
                    an= 8'b1110_1111; show=PC[3:0];
                end
            else if(cnt_show<=13'd4999)begin
                    an= 8'b1111_1011; show=5'b10000;  //r
                end
            else begin
                    an= 8'b1111_1101; show=5'b10010; //U
                end
end
```

```
        else begin  //输出寄存器内容
            led16=3'b010;
            led17=3'b010;
            if(cnt_show<=13'd999)begin
                     an= 8'b0111_1111;    show=R[num_reg_show][15:12];//
                end
            else if(cnt_show<=13'd1999)begin
                     an= 8'b1011_1111;    show=R[num_reg_show][11:8];//num_reg_show;
                end
            else if(cnt_show<=13'd2999)begin
                     an= 8'b1101_1111;   show=R[num_reg_show][7:4];
                end
            else if(cnt_show<=13'd3999)begin
                     an= 8'b1110_1111;   show=R[num_reg_show][3:0];
                end
            else if(cnt_show<=13'd4999)begin
                     an= 8'b1111_1101;   show=5'b10000;   //r
                end
            else begin
                     an= 8'b1111_1110;   show=num_reg_show;
                end
        end
end
```

上下图分别是在七段数码管上显示寄存器、内存单元的部分控制代码。

```
            1:begin   //输出内存
                   led16=3'b001;
                   led17=3'b001;
                  if(cnt_show<=13'd999)begin
                             an= 8'b0111_1111;    show=memory[address_memory_show][15:12];
                             end
                  else if(cnt_show<=13'd1999)begin
                             an= 8'b1011_1111;    show=memory[address_memory_show][11:8];  //F
                             end
                   else if(cnt_show<=13'd2999)begin
                             an= 8'b1101_1111;    show=memory[address_memory_show][7:4];
                             end
                   else if(cnt_show<=13'd3999)begin
                             an= 8'b1110_1111;    show=memory[address_memory_show][3:0];
                             end
                   else if(cnt_show<=13'd4999)begin
                             an= 8'b1111_1101;   show=address_memory_show[7:4];
                             end
                   else begin
                             an= 8'b1111_1110;   show=address_memory_show[3:0];
                             end
               end
          endcase
    end
    segment_decoder sd(show,digit_x);
endmodule
```

三．实验心得与总结

（1）确定题目及开始实验前的规划

经过一学期数电实验课程的学习，我对 Vivado 环境、Verilog 语法都有了一定的了解，对有限状态机的控制逻辑也有了自己的体会。而在《计算机系统概论》课程中，我接触到了基于冯诺依曼架构的小计算机 LC-3，负责《计算机系统概论》教学的安虹老师也多次鼓励我们自己动手实现 LC-3。我想本次大作业正是巩固自己所学、深入对计算机体系结构的了解的好机会。因此我确定大作业题目是 LC-3 FPGA 实现。

开始实验前，我仔细阅读了 LC-3 的数据通路（图 1）及 LC-3 有限状态机（图 2），大致规划了需要实现的功能部件及有限状态机的控制逻辑。并决定使用 7 段数码管和 led 灯显示结果、用开关输入指令。让 LC-3 能够在两种控制模式之间切换的想法是在后续实验过程中产生的。

（2）实验中的坎坷与喜悦

1. 控制逻辑的精简：

实现数据通路过程中，由于控制信号很多，程序写起来很复杂。初始时，我在代码中设置了多个 always 块，并定义总线为 reg 型。但是这样在 Vivado 实现过程中出现了 Multiple Driven 报错，原因是总线作为 reg 类型在多个 always 块中赋值。并且初始时我将 LC-3 功能模块分得过细，下载到板子上执行结果不理想，原因是各模块之间控制

信号冲突。经过仔细考虑，我对控制逻辑进行了精简，最后决定仿效lab10，只使用 3 个 always 块分别负责状态转换、状态内容及显示输出。精简之后，控制信号减少了，程序的逻辑更清晰。

2. LC-3 工作模式的改进：

最初，我的指令只能从开关读入，而不能从内存读取，原因是找不到在工作时切换控制模式的方法。在与同学的交流和自己的摸索之下，我使用了 LC-3 指令集中预留的指令 1101，实现了当开关输入 1101_111 时工作模式切换的效果。同时 1101 也被我用来作为查看 LC-3 寄存器、内存内容的指令，对 1101 指令功能的拓展是我在设计 LC-3 FPGA 实现过程中的一个关键创新。

3. 显示输出的改进：

在同学的启发之下，我使用了 RGB led 灯，当使用 1101 指令分别查看寄存器文件 R0 至 R7、内存内容、PC、切换工作模式时，RGB led 灯会分别亮绿、蓝、红、黄色，有助于区分当前查看的是什么内容，增添了视觉效果。

本次实验我先后写了 3 个版本，课余时间不断调试，力求功能上的完整。经过一周艰苦的努力，终于达到了基本实现 LC-3 功能的目标，这让我非常喜悦，感到这一周来的努力没有白费。

（3）实验后的反思与收获

1. 开展较大设计项目（如本次大作业）之前应当仔细规划、综合考虑。否则在过程中容易走错方向，造成时间的浪费。

因为没有考虑好就开始写，我写的第一个版本出现了很多设计上的问题，如在多个 always 里面对一个 reg 赋值，出现报错。只能重新设计代码结构，造成时间浪费。

2. 实验中遇到问题需要认真动脑分析，而不是碰运气式修改。并且为防止疏漏，在修正一处错误之后还需要检查一下全局。

过程中有几次实现时报错，发现是变量名写错了，匆匆修改后再次实现还是报错，仔细查看代码后才发现有多处变量名需要修改，结果自己只改了其中一处。

3. 有时遇到自己难以解决的问题不要钻牛角尖，与同学交流或者询问老师、助教是更好的办法。

实验中我遇到了一个加法信号控制上的问题，自己反复调试但加法结果怎么也不对，在同学的帮助下我才发现了问题所在。