

LC-3 Simulator

PB18111789 夏寒

LC-3介绍

LC-3出自《计算机系统概论(第二版)》一书，是一个图灵完备的16位计算机，内置了 $2^{10} \times 16$ bit的内存和8个16位的寄存器。

指令集及每条指令的具体解释如下：

指令	功能
ADD	相加
AND	按位与
BR	根据条件码跳转
JMP	无条件跳转
JSR/JSRR	将当前的PC存入R7，然后跳转
LD	加载内存数据，采用PC相对寻址
LDI	加载内存数据，采用间接寻址
LDR	加载内存数据，采用寄存器寻址
LEA	将PC+PCoffset存入寄存器
NOT	按位取非
RET	将R7的值赋给PC
RTI	中断返回
ST	将指定寄存器的内容存入内存，采用PC相对寻址
STI	将指定寄存器的内容存入内存，采用间接寻址
STR	将指定寄存器的内容存入内存，采用寄存器寻址
TRAP	将内存中地址为trapvect的数据加载到PC中
HALT	相当于 TRAP x25

操作说明：

为了为下文的介绍做铺垫，先介绍一些理解交互操作所需的程序变量：

变量名	含义
内存地址	数据在内存中的地址，范围 $0 \sim 2^{10}-1$
内存内容	内存中存储的内容，在LC-3的实现中不严格区分指令和数据
PC	program counter，指向当前执行的指令在内存中的地址
cur	光标(cursor)，表示当前被数码管显示的内存地址
cnt	counter，用来计数执行的指令条数
mod	交互模式状态代码
state	内核状态机状态代码

注：其实整个LC-3中只有一个有限状态机，但为了方便理解和编写代码，将状态机的状态代码分为了 mod 和 state 两部分，这种表述与只用一个状态变量是等价的。

下面进行详细介绍。

为了方便与用户的交互，LC-3 Simulator总共设计了4种模式：

模式	功能	状态代码(mod)
Write	写入	00
Jump	跳转	10
Run	运行	01
(Undefined)	(未定义)	11

LC-3 的内核，也就是执行程序的核心部分，使用有限状态自动机进行描述。共有32种状态，大部分状态与具体的指令逻辑有关，不做特别介绍，可以参见代码。下面说明一些关键的状态：

state	描述
0	停机状态
1	将内存的写使能(write enable)信号置为0，同时cnt加一
2	取指令，PC加一，同时检查PC是否为非法值，如果是，跳转到状态0(停机)

显示部分使用的是板载的数码管和LED灯：

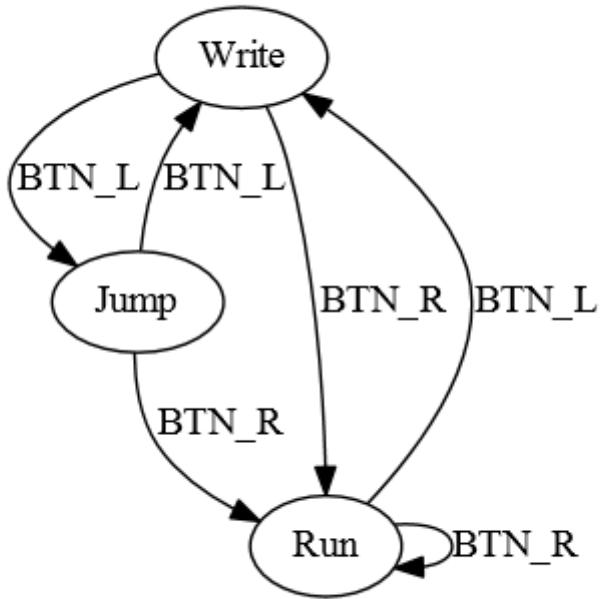
- 数码管
 - 始终显示的是当前光标所在行的地址和内容，二者均使用16进制表示。
- LED灯
 - 在Write状态下
 - 显示的是PC值(这样设置主要是为了方便调试)；
 - 在其他状态下
 - 左侧9位显示的是指令执行总条数的数值(cur)
 - 中间5位显示的是当前的LC-3内核自动机的状态代码(state)
 - 右边两位显示的是当前的交互模式代码(mod)

可供操作的有五个按钮和16个开关。

可以使用上下左右中5个按钮进行操控：

按钮	作用
上 (BTN_U)	在 write 和 jump 模式下用来使 cur 减一
下 (BTN_D)	在 write 和 jump 模式下用来使 cur 加一
左 (BTN_L)	用来在 write 和 jump 模式间相互切换，或强制终止运行(Run->write)
右 (BTN_R)	将 sw 赋值给 PC，并开始执行，切换到 Run 模式
中 (BTN_C)	确定键，在 write 模式下将 sw 赋值给 cur 指向的内存空间，在 jump 模式下将 sw 赋值给 cur

其状态转移图如下：



测试

为了方便测试，在FPGA中内置了求解最大公因数的程序。

以下是其对应的LC-3汇编程序描述：

```
.ORIG x0200
LDI R0,A
LDI R1,B
AND R0,R0,R0
BRp #2
NOT R0,R0
ADD R0,R0,#1
AND R1,R1,R1
BRn #2
NOT R1,R1
ADD R1,R1,#1
TEST ADD R2,R1,R0
BRz ANS
BRn ELSE
AND R2,R1,R1
TEST1 ADD R3,R2,R0
ADD R3,R3,R2
BRp CASE1
ADD R0,R0,R2
BRnzp TEST
CASE1 ADD R2,R2,R2
BRnzp TEST1
ELSE AND R2,R0,R0
TEST2 ADD R3,R2,R1
ADD R3,R2,R3
BRn CASE2
ADD R1,R1,R2
BRnzp TEST
```

```

CASE2    ADD      R2 ,R2 ,R2
          BRnzp   TEST2
ANS      STI      R0 ,C
          HALT

A       .FILL    x0300
B       .FILL    x0301
C       .FILL    x0302
          .END

```

对其中出现的伪指令的解释：

伪指令	作用
.ORIG	声明程序起始地址
.FILL	直接填充数据。例如 .FILL x0300 表示该行的内容为 x0300
.END	表示程序到此为止

对应的二进制码为：

```

0000001000000000
1010000000011110
1010001000011110
0101000000000000
0000001000000010
1001000000111111
0001000000100001
0101001001000001
0000100000000010
1001001001111111
0001001001100001
0001010001000000
00000100000010001
0000100000001000
0101010001000001
0001011010000000
00010110110000010
00000010000000010
0001000000000010
000011111110111
0001010010000010
000011111111001
0101010000000000
0001011010000001
0001011010000011
0000100000000010
0001001001000010
0000111111101111
0001010010000010
000011111111001
1011000000000011
1111000000100101
0000001100000000
0000001100000001
00000011000000010

```

测试数据:

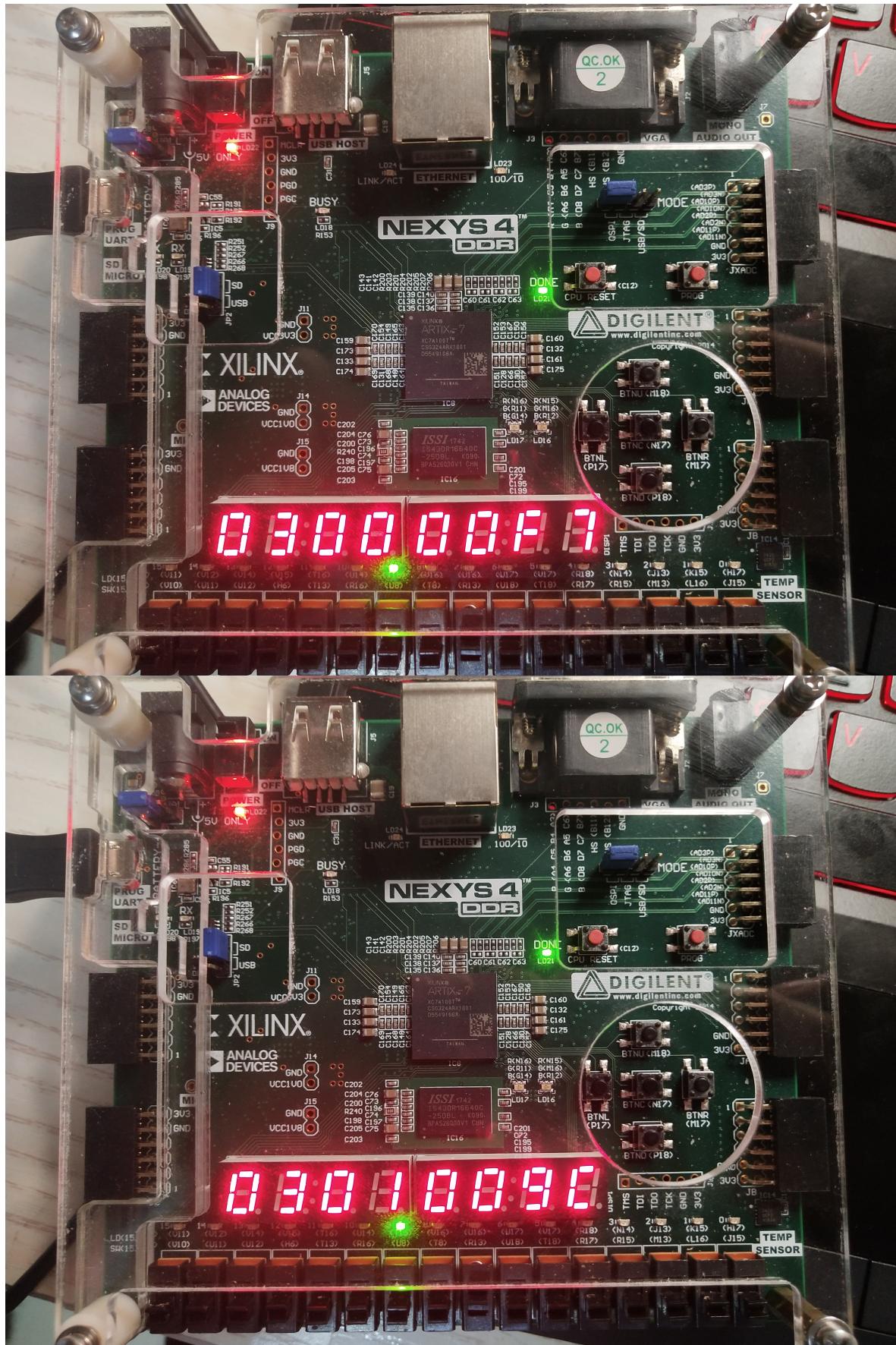
000000001110111 ($x00F7 = 13 * 19$)

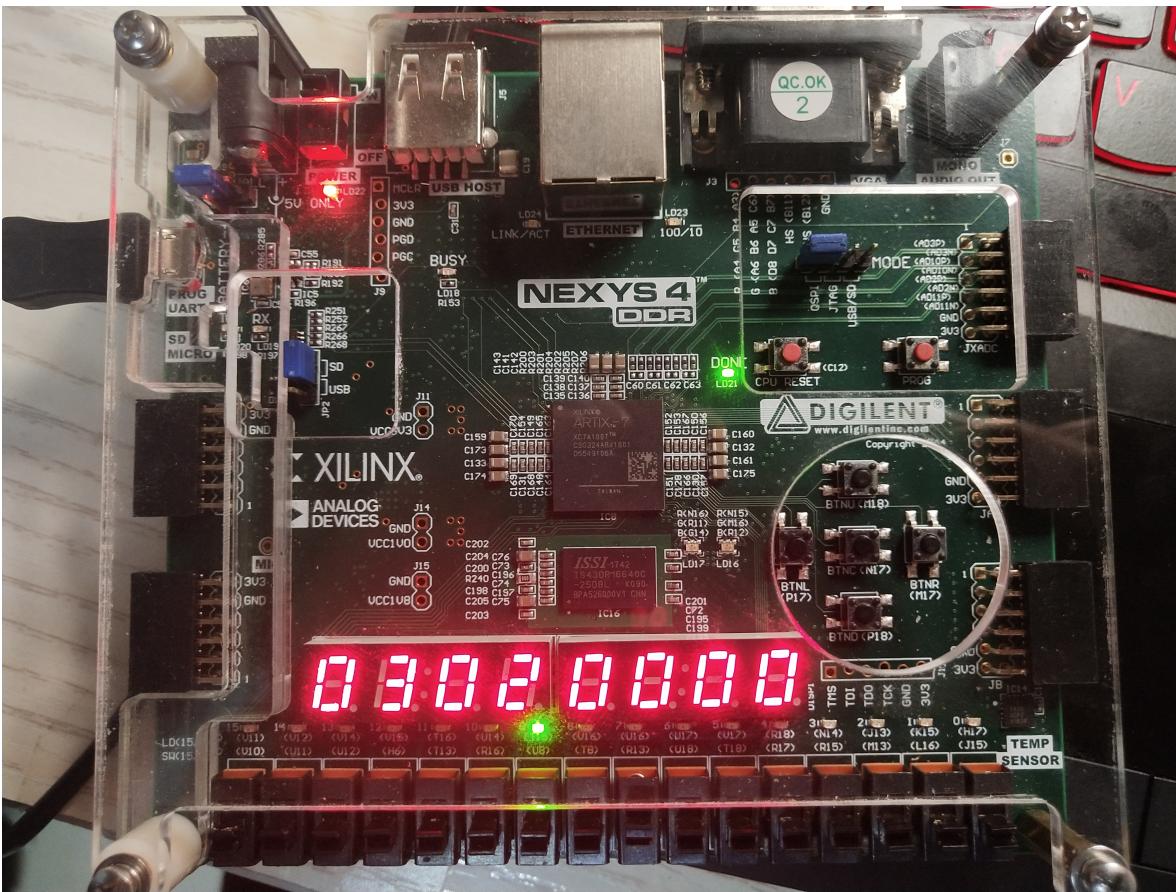
0000000010011100 ($x009C = 13 * 12$)

预期结果:

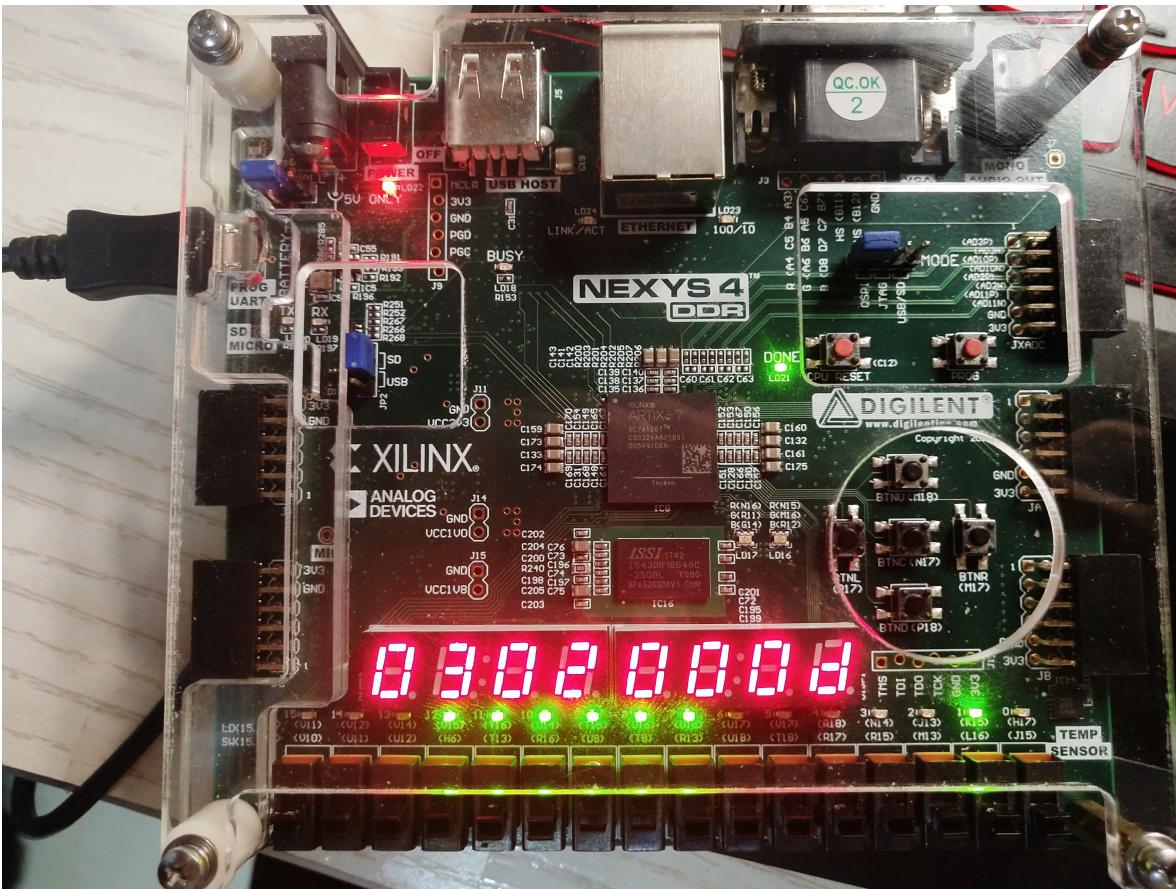
00000000000001101 ($x000d = 13$)

测试照片:





运行后：



同时可以看到总共执行了63条指令。

该程序经过了算法优化，最恶心刁钻的测试数据也需要大约600条指令，平均执行指令数大约200条，在没有右移和取余指令的情况下基本上达到了优化的极限。

代码解释

按键去抖动并取上升沿的模块

为了使得能在时钟上升沿时btn_clean能被采样到有且只有一次，在该模块中输出的btn_clean的信号起始和终止都在时钟的下降沿，且时长为一个时钟周期。

```
module clean_posedge(
    input clk,
    input btn,
    output btn_clean);
    reg [15:0] cnt;
    reg [1:0] delay_sig;

    always @(negedge clk)
    begin
        if(btn == 1'b0)
            cnt <= 16'b0;
        else if(cnt < 16'h8000)
            cnt <= cnt +1'b1;
    end
    always @(negedge clk)
        delay_sig <= {delay_sig[0],cnt[15]};

    assign btn_clean = delay_sig[0] & (~delay_sig[1]);
endmodule
```

IC-3主機板

LC-3的主要逻辑

具体部分的代码解释见注释

```
module lc3c
    input          CLK      ,
    input [15:0]   SW       ,
    input [4:0]    BTN      ,
    output reg [7:0] SSEG_CA ,
    output reg [7:0] SSEG_AN ,
    output [15:0]  LED      ;
);

//////////////////////////////产生25MHz的时钟供内核使用
//////////////////////////////LC-3 控制
//////////////////////////////内核
// 产生25MHz的时钟供内核使用
reg [1:0] clk_25m_cnt;
wire clk_25m;

always @(posedge CLK)
begin
    clk_25m_cnt <= clk_25m_cnt + 1;
end

assign clk_25m = clk_25m_cnt[1];

//////////////////////////////LC-3 控制
//////////////////////////////内核
// LC-3 控制
// 内核
reg [15:0] PC;
reg [15:0] IR;
```

```

reg      [2:0]  CC;      //N Z P
reg      [15:0] CCR;     //用于计算CC的中间寄存器
wire     BEN;
reg      [4:0]  state;   //LC-3内核自动机的状态，共有32个
//内存
reg      [9:0]  MAR;
reg      [15:0] MDR_in;
wire     [15:0] MDR_out;
reg      [15:0] MDR;     //缓冲用，防止输入输出直接相接
reg      M_WE;
//OS
reg      [15:0] cur;    //光标
reg      [1:0]  mod;    //jump:2    run:1  halt/write:0
wire     [4:0]  BTN_clean;
reg      [8:0]  ins_cnt;//指令计数器

///////////////////////////////
// LC-3 寄存器
/////////////////////////////
wire      [2:0]  DR;     //二进制表示
wire      [2:0]  SR1;
wire      [2:0]  SR2;

assign    DR = IR[11:9];
assign    SR1 = IR[8:6];
assign    SR2 = IR[2:0];

reg      [15:0] R[7:0];

///////////////////////////////
// LC-3 内存
/////////////////////////////
parameter memory_range = 1 << 10;
dist_mem_gen_0 M(
.a      (MAR),
.d      (MDR_in),
.spo   (MDR_out),
.clk   (CLK),
.we    (M_WE));

///////////////////////////////
// LC-3 内核
/////////////////////////////
assign  BEN = (IR[11] & CC[2]) |
        (IR[10] & CC[1]) |
        (IR[9] & CC[0]);
always @(*)
begin
  if(CCR == 16'b0)
    CC <= 3'b010;
  else if(CCR[15] == 1'b1)
    CC <= 3'b100;
  else
    CC <= 3'b001;
end

always @(`posedge clk_25m)

```

```

begin
  case(state)
    5'd1  :
    begin
      M_WE <= 1'b0;           //暂时禁止写入，防止在state == 2时内存被修改
      ins_cnt <= ins_cnt + 1;
      state <= 5'd2;
    end
    5'd2  :
    begin
      if(PC >= memory_range || mod == 2'b0) //PC越界或强制停机
        begin
          mod <= 2'd0;
          state <= 5'd0;
        end
      else
        begin
          MAR <= PC[9:0];
          PC <= PC + 1;
          state <= 5'd3;
        end
    end
    5'd3  :
    begin
      IR <= MDR_out;
      state <= 5'd4;
    end
    5'd4  :
    begin
      case(IR[15:12])
        4'b0001:
          state <= 5'd5; //ADD
        4'b0101:
          state <= 5'd6; //AND
        4'b1001:
          state <= 5'd7; //NOT
        4'b1111:
          state <= 5'd8; //TRAP
        4'b1110:
          state <= 5'd11; //LEA
        4'b0010:
          state <= 5'd12; //LD
        4'b0110:
          state <= 5'd13; //LDR
        4'b1010:
          state <= 5'd18; //LDI
        4'b1011:
          state <= 5'd19; //STI
        4'b0111:
          state <= 5'd22; //STR
        4'b0011:
          state <= 5'd23; //ST
        4'b0100:
          state <= 5'd29; //JSR
        4'b1100:
          state <= 5'd28; //JMP
        4'b0000:
          state <= 5'd26; //BR
      endcase
    end
  endcase
end

```

```

        endcase
    end
5'd5  :
//ADD
begin
    if(IR[5])
    begin
        R[DR] <= R[SR1] + {IR[4]?11'b111_1111_1111:11'b0,IR[4:0]};
        CCR <= R[SR1] + {IR[4]?11'b111_1111_1111:11'b0,IR[4:0]};
    end
    else
    begin
        R[DR] <= R[SR1] + R[SR2];
        CCR <= R[SR1] + R[SR2];
    end
    state <= 5'd1;
end
5'd6  :
//AND
begin
    if(IR[5])
    begin
        R[DR] <= R[SR1] & {IR[4]?11'b111_1111_1111:11'b0,IR[4:0]};
        CCR <= R[SR1] & {IR[4]?11'b111_1111_1111:11'b0,IR[4:0]};
    end
    else
    begin
        R[DR] <= R[SR1] & R[SR2];
        CCR <= R[SR1] & R[SR2];
    end
    state <= 5'd1;
end
5'd7  :
//NOT
begin
    R[DR] <= ~R[SR1];
    CCR <= ~R[SR1];
    state <= 5'd1;
end
5'd8  :
//TRAP
begin
    MAR <= {IR[7]?2'b11:2'b0,IR[7:0]};
    state <= 5'd9;
end
5'd9  :
begin
    R[7] <= PC;
    state <= 5'd10;
end
5'd10 :
begin
    PC <= MDR_out;
    state <= 5'd1;
end
5'd11 :
//LEA
begin

```

```

        R[DR] <= PC + {IR[8]?7'b111_1111:7'b0,IR[8:0]};
        CCR <= PC + {IR[8]?7'b111_1111:7'b0,IR[8:0]};
        state <= 5'd1;
    end
    5'd12 :
    //LD
    begin
        MAR <= PC + {IR[8]?7'b111_1111:7'b0,IR[8:0]};
        state <= 5'd14;
    end
    5'd13 :
    //LDR
    begin
        MAR <= R[SR1] + {IR[5]?10'b11_1111_1111:10'b0,IR[5:0]};
        state <= 5'd14;
    end
    5'd14 :
    begin
        state <= 5'd15;
    end
    5'd15 :
    begin
        R[DR] <= MDR_out;
        CCR <= MDR_out;
        state <= 5'd1;
    end
    5'd16 :
    begin
        MAR <= MDR[9:0];
        state <= 5'd14;
    end
    5'd17 :
    begin
        MDR <= MDR_out;
        state <= 5'd16;
    end
    5'd18 :
    begin
        MAR <= PC + {IR[8]?7'b111_1111:7'b0,IR[8:0]};
        state <= 5'd17;
    end
    5'd19 :
    begin
        MAR <= PC + {IR[8]?7'b111_1111:7'b0,IR[8:0]};
        state <= 5'd20;
    end
    5'd20 :
    begin
        MDR <= MDR_out;
        state <= 5'd21;
    end
    5'd21 :
    begin
        MAR <= MDR[9:0];
        state <= 5'd24;
    end
    5'd22 :
    //STR

```

```

begin
    MAR <= R[SR1] + {IR[5]?10'b11_1111_1111:10'b0,IR[5:0]};
    state <= 5'd24;
end
5'd23 :
begin
    MAR <= PC + {IR[5]?10'b11_1111_1111:10'b0,IR[8:0]};
    state <= 5'd24;
end
5'd24 :
begin
    MDR_in <= R[DR];
    state <= 5'd25;
end
5'd25 :
begin
    M_WE <= 1'b1;
    state <= 5'd1;
end
5'd26 :
begin
    if(BEN)
        state <= 5'd27; //BR 1
    else
        state <= 5'd1; //BR 0
end
5'd27 :
begin
    PC <= PC + {IR[8] ? 7'b111_1111 : 7'b0,IR[8:0]};
    state <= 5'd1;
end
5'd28 :
begin
    PC <= R[SR1];
    state <= 5'd1;
end
5'd29 :
//JSR/JSR
begin
    R[7] <= PC;
    if(IR[11])
        state <= 5'd30;
    else
        state <= 5'd31;
end
5'd30 :
begin
    PC <= PC + {IR[10]?5'b11111:5'b0,IR[10:0]};
    state <= 5'd1;
end
5'd31 :
begin
    PC <= R[SR1];
    state <= 5'd1;
end
default:
//state == 0 , 当执行HALT后跳转到该状态
begin

```

```

    MAR <= cur[9:0];
    if(mod == 2'b01)
    begin
        PC <= SW;           //启动, 从SW开始执行
        ins_cnt <= 0;       //指令计数器清空
        state <= 5'd1;
    end
    else
        state <= 5'd0;      //保持停机状态
    if (mod == 2'b00 && BTN_clean[4])      //Press Enter
    begin
        M_WE <= 1'b1;     //允许写入
        MDR_in <= SW;
    end
    else
        M_WE <= 1'b0;
end
endcase

// LC-3 OS
case (mod)
    2'd0:
        //Write
    begin
        if (BTN_clean[0])
        begin
            if (cur != 0)
                cur <= cur - 1;
        end
        else if (BTN_clean[1])
        begin
            mod <= 2'd2;
        end
        else if (BTN_clean[2])
        begin
            mod <= 2'd1;
        end
        else if (BTN_clean[3])
        begin
            if (cur != memory_range - 1)
                cur <= cur + 1;
        end
    end
    2'd1:
        //Run
    begin
        if (BTN_clean[1])
            mod <= 2'd0;
    end
    2'd2:
        //Jump
    begin
        if (BTN_clean[0])
        begin
            if (cur != 0)
                cur <= cur - 1;
        end
        else if (BTN_clean[1])

```

```

        mod <= 2'd0;
    else if (BTN_clean[2])
        mod <= 2'd1;
    else if (BTN_clean[3])
    begin
        if (cur != memory_range - 1)
            cur <= cur + 1;
    end
    else if (BTN_clean[4])
        cur <= {6'b0,SW[9:0]}; //跳转到SW对应的地址
    end
    default:
    begin
        mod <= 2'd0;
    end
endcase
end

///////////////////////////////
// LC-3 交互
///////////////////////////////

initial
begin
    cur = 16'h200;
    state = 5'd0;
    mod = 2'd0;
    ins_cnt = 9'b0;
    PC = 10'h200;
    M_WE = 1'b0;
    R[0] = 16'b0;
    R[1] = 16'b0;
    R[2] = 16'b0;
    R[3] = 16'b0;
    R[4] = 16'b0;
    R[5] = 16'b0;
    R[6] = 16'b0;
    R[7] = 16'b0;
end

assign LED = (mod == 2'b0) ? PC : {ins_cnt,state,mod};

clean_posedge cl_p_0(clk_25m,BTN[0],BTN_clean[0]);
clean_posedge cl_p_1(clk_25m,BTN[1],BTN_clean[1]);
clean_posedge cl_p_2(clk_25m,BTN[2],BTN_clean[2]);
clean_posedge cl_p_3(clk_25m,BTN[3],BTN_clean[3]);
clean_posedge cl_p_4(clk_25m,BTN[4],BTN_clean[4]);

///////////////////////////////
// LC-3 显示
/////////////////////////////
wire [7:0] digit0,digit1,digit2,digit3,digit4,digit5,digit6,digit7; //显示用数码
wire [3:0] num0,num1,num2,num3,num4,num5,num6,num7; //显示的数学值
reg [16:0] cnt;

always @ (posedge CLK)

```

```

begin
    cnt <= cnt + 1;
    case (cnt[16:14])
        3'b000:
            begin
                SSEG_AN <= 8'b01111111;
                SSEG_CA <= digit0;
            end
        3'b001:
            begin
                SSEG_AN <= 8'b10111111;
                SSEG_CA <= digit1;
            end
        3'b010:
            begin
                SSEG_AN <= 8'b11011111;
                SSEG_CA <= digit2;
            end
        3'b011:
            begin
                SSEG_AN <= 8'b11101111;
                SSEG_CA <= digit3;
            end
        3'b100:
            begin
                SSEG_AN <= 8'b11110111;
                SSEG_CA <= digit4;
            end
        3'b101:
            begin
                SSEG_AN <= 8'b11111011;
                SSEG_CA <= digit5;
            end
        3'b110:
            begin
                SSEG_AN <= 8'b11111101;
                SSEG_CA <= digit6;
            end
        3'b111:
            begin
                SSEG_AN <= 8'b11111110;
                SSEG_CA <= digit7;
            end
    endcase
end
assign num0 = cur[15:12];
assign num1 = cur[11:8];
assign num2 = cur[7:4];
assign num3 = cur[3:0];
assign num4 = MDR_out[15:12];
assign num5 = MDR_out[11:8];
assign num6 = MDR_out[7:4];
assign num7 = MDR_out[3:0];
dist_mem_gen_1 dist_mem_gen_0(
    .a      (num0),
    .spo    (digit0));
dist_mem_gen_1 dist_mem_gen_1(
    .a      (num1),

```

```
.spo    (digit1));
dist_mem_gen_1 dist_mem_gen_2(
.a      (num2),
.spo    (digit2));
dist_mem_gen_1 dist_mem_gen_3(
.a      (num3),
.spo    (digit3));
dist_mem_gen_1 dist_mem_gen_4(
.a      (num4),
.spo    (digit4));
dist_mem_gen_1 dist_mem_gen_5(
.a      (num5),
.spo    (digit5));
dist_mem_gen_1 dist_mem_gen_6(
.a      (num6),
.spo    (digit6));
dist_mem_gen_1 dist_mem_gen_7(
.a      (num7),
.spo    (digit7));
endmodule
```

总结

之前参加ICS课程就一直想在硬件上实现LC-3，现在终于在FPGA上完成了。

本人实现的LC-3相对于《计算机系统概论》中描述的还是有一定的区别：

- 内存空间设置成了 2^{10} 个地址，而不是书中的 2^{16} ，主要的考虑是如果内存过大，在VIVADO生成的时候会耗费大量的时间，而内存中的很多空间是根本不会用到的，此外缩小内存空间不会产生本质上的区别。
- 删去了有关中断处理的部分，因为没有连接外设，所以不会用到中断机制，留着实属鸡肋。
- 停机操作的触发条件是PC为非法值，而不是修改控制寄存器内容使时钟停止。
- 内存中TRAP矢量表x0025处的内容改为了xFFFF，这样在执行HALT指令时可以直接将PC设为非法值从而触发停机。
- TRAP矢量表中其他地址的内容都指向了非法值。

代码已经上传到本人的GitHub上，[项目地址](#)。求star

另外LC-3汇编代码可以使用本人自行编写的汇编器转化为二进制码，[项目地址](#)。