程式說明

Nexys4.xdc，將原本的 sw15 改成 rst，作為 reset 用途

```
set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 } [get_ports { li0 }]; #IO_L24N_T3_RS0_15 Sch=sw[0]
set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 } [get_ports { li1 }]; #IO_L3N_T0_DQS_EMCCLK_14 Sch=sw[1]
set_property -dict { PACKAGE_PIN M13   IOSTANDARD LVCMOS33 } [get_ports { li2 }]; #IO_L6N_T0_D08_VREF_14 Sch=sw[2]
set_property -dict { PACKAGE_PIN R15   IOSTANDARD LVCMOS33 } [get_ports { li3 }]; #IO_L13N_T2_MRCC_14 Sch=sw[3]
set_property -dict { PACKAGE_PIN R17   IOSTANDARD LVCMOS33 } [get_ports { li4 }]; #IO_L12N_T1_MRCC_14 Sch=sw[4]
set_property -dict { PACKAGE_PIN T18   IOSTANDARD LVCMOS33 } [get_ports { li5 }]; #IO_L7N_T1_D10_14 Sch=sw[5]
set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [get_ports { li6 }]; #IO_L17N_T2_A13_D29_14 Sch=sw[6]
set_property -dict { PACKAGE_PIN R13   IOSTANDARD LVCMOS33 } [get_ports { li7 }]; #IO_L5N_T0_D07_14 Sch=sw[7]
set_property -dict { PACKAGE_PIN T8    IOSTANDARD LVCMOS18 } [get_ports { li8 }]; #IO_L24N_T3_34 Sch=sw[8]
set_property -dict { PACKAGE_PIN U8    IOSTANDARD LVCMOS18 } [get_ports { li9 }]; #IO_25_34 Sch=sw[9]
set_property -dict { PACKAGE_PIN R16   IOSTANDARD LVCMOS33 } [get_ports { li10 }]; #IO_L15P_T2_DQS_RDWR_B_14 Sch=sw[10]
set_property -dict { PACKAGE_PIN T13   IOSTANDARD LVCMOS33 } [get_ports { li11 }]; #IO_L23P_T3_A03_D19_14 Sch=sw[11]
set_property -dict { PACKAGE_PIN H6    IOSTANDARD LVCMOS33 } [get_ports { li12 }]; #IO_L24P_T3_35 Sch=sw[12]
#set_property -dict { PACKAGE_PIN U12   IOSTANDARD LVCMOS33 } [get_ports { sw13 }]; #IO_L20P_T3_A08_D24_14 Sch=sw[13]
#set_property -dict { PACKAGE_PIN U11   IOSTANDARD LVCMOS33 } [get_ports { SW[14] }]; #IO_L19N_T3_A09_D25_VREF_14 Sch=sw[14]
set_property -dict { PACKAGE_PIN V10   IOSTANDARD LVCMOS33 } [get_ports { rst }]; #IO_L21P_T3_DQS_14 Sch=sw[15]
```

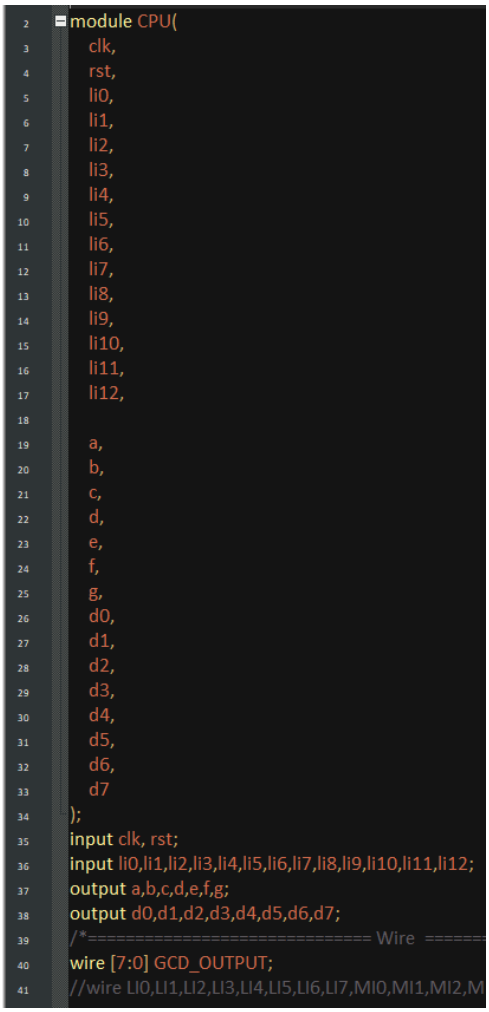
IF.v 將原本 testbench 裡的 cpu.IF.instruction 放進 IF 裡面，並且把初始值用 resgister 來存，不使用 sw、lw 等指令，改用 add 代替

```
always @(posedge clk or posedge rst)
begin
  if(rst) begin
    IR <= 32'd0;
    instruction[ 0] = 32'b000000_00101_00001_01011_00000_100100;                    //and $11, $5, $1  if $10=0 ,input is even;   if $10=1, input is odd
    instruction[ 1] = 32'b000000_00001_00101_00111_00000_100000;                    // add $7, $1, $5              // $7 store input+1
    instruction[ 2] = 32'b000000_00000_00000_00000_00000_100000;  //NOP(add $0, $0, $0)
    instruction[ 3] = 32'b000000_00000_00000_00000_00000_100000;  //NOP(add $0, $0, $0)
    instruction[ 4] = 32'b000100_01011_00000_0000_0000_0000_1000;                   //beq, $11, $0, start         //beq go down 8
    instruction[ 5] = 32'b000000_00000_00000_00000_00000_100000;  //NOP(add $0, $0, $0)
    instruction[ 6] = 32'b000000_00000_00000_00000_00000_100000;  //NOP(add $0, $0, $0)
    instruction[ 7] = 32'b000000_00000_00000_00000_00000_100000;  //NOP(add $0, $0, $0)
    instruction[ 8] = 32'b000000_00111_00001_00111_00000_100000;                    // add $7, $7, $1          // $7 +1
```

ID.v 將 register 的初始值放在這裡，指播的運算放在 reg[5]中，其他的依照程式需求放進初始值

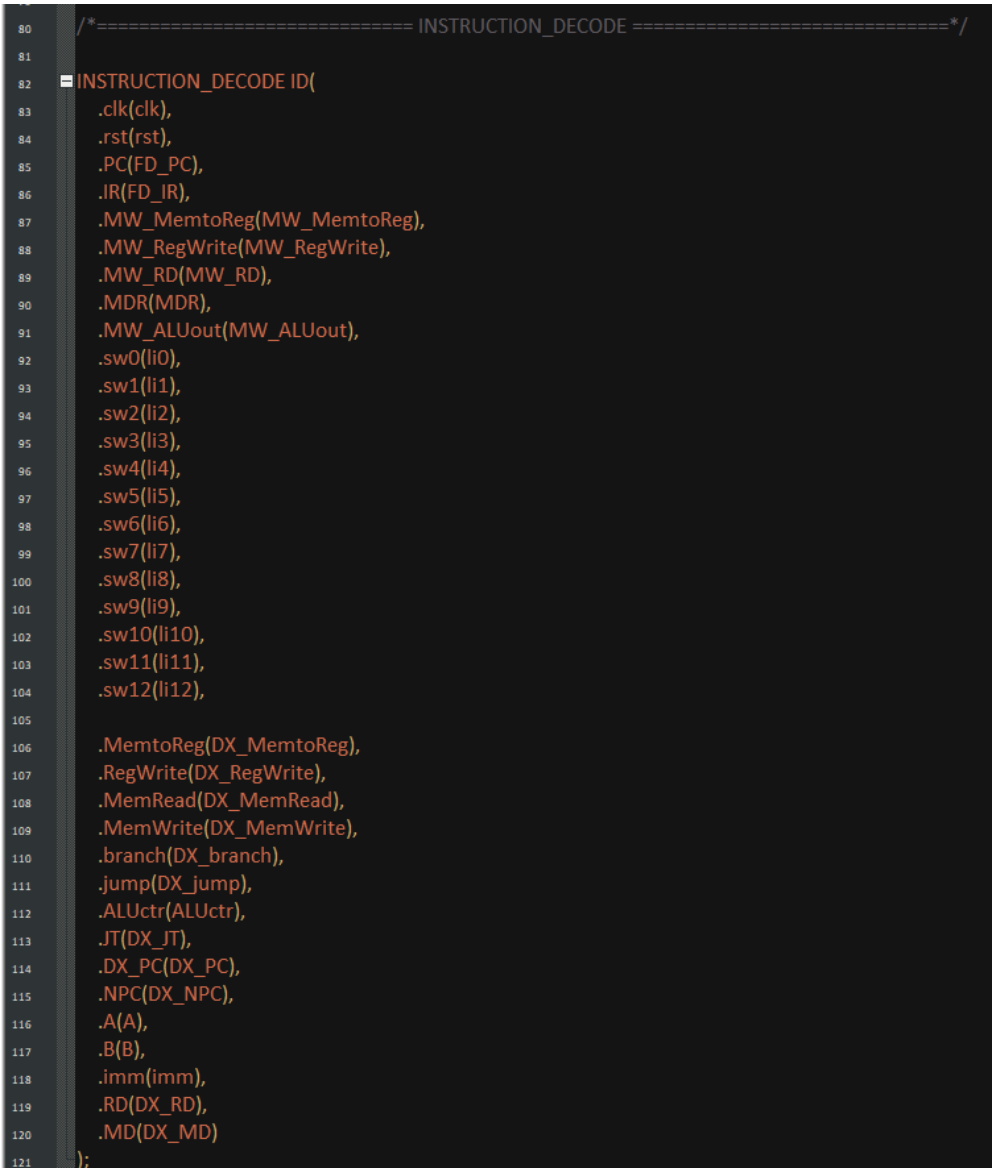
```
    REG[0] <= 32'b00000000000000000000000000000000;
    REG[1] <= 32'b00000000000000000000000000000001;
    REG[2] <= 32'b00000000000000000000000000000010;
    REG[3] <= 32'b00000000000000000000000000000011;
    REG[4] <= 32'b00000000000000000000000000000101;
    REG[5] <= {19'd0,sw12,sw11,sw10,sw9,sw8,sw7,sw6,sw5,sw4,sw3,sw2,sw1,sw0};
    REG[6] <= 32'b00000000000000000000000000000000;
    REG[7] <= 32'b00000000000000000000000000000000;
    REG[8] <= 32'b00000000000000000000000000000000;
    REG[9] <= 32'b00000000000000000000000000000000;
    REG[10] <= 32'b00000000000000000000000000000000;
    REG[11] <= 32'b00000000000000000000000000000000;
    REG[12] <= 32'b00000000000000000000000000000000;
    REG[13] <= 32'b00000000000000000000000000000000;
    REG[14] <= 32'b00000000000000000000000000000000;
    REG[15] <= 32'b00000000000000000000000000000000;
```

EXE.v MEM.v 沒有修改，CPU.v 放入在 Nexys4.xdc 裡的指播名稱



```
module CPU(
    clk,
    rst,
    li0,
    li1,
    li2,
    li3,
    li4,
    li5,
    li6,
    li7,
    li8,
    li9,
    li10,
    li11,
    li12,

    a,
    b,
    c,
    d,
    e,
    f,
    g,
    d0,
    d1,
    d2,
    d3,
    d4,
    d5,
    d6,
    d7
);
input clk, rst;
input li0,li1,li2,li3,li4,li5,li6,li7,li8,li9,li10,li11,li12;
output a,b,c,d,e,f,g;
output d0,d1,d2,d3,d4,d5,d6,d7;
/*============================ Wire  =======
wire [7:0] GCD_OUTPUT;
//wire LI0,LI1,LI2,LI3,LI4,LI5,LI6,LI7,MI0,MI1,MI2,M
```

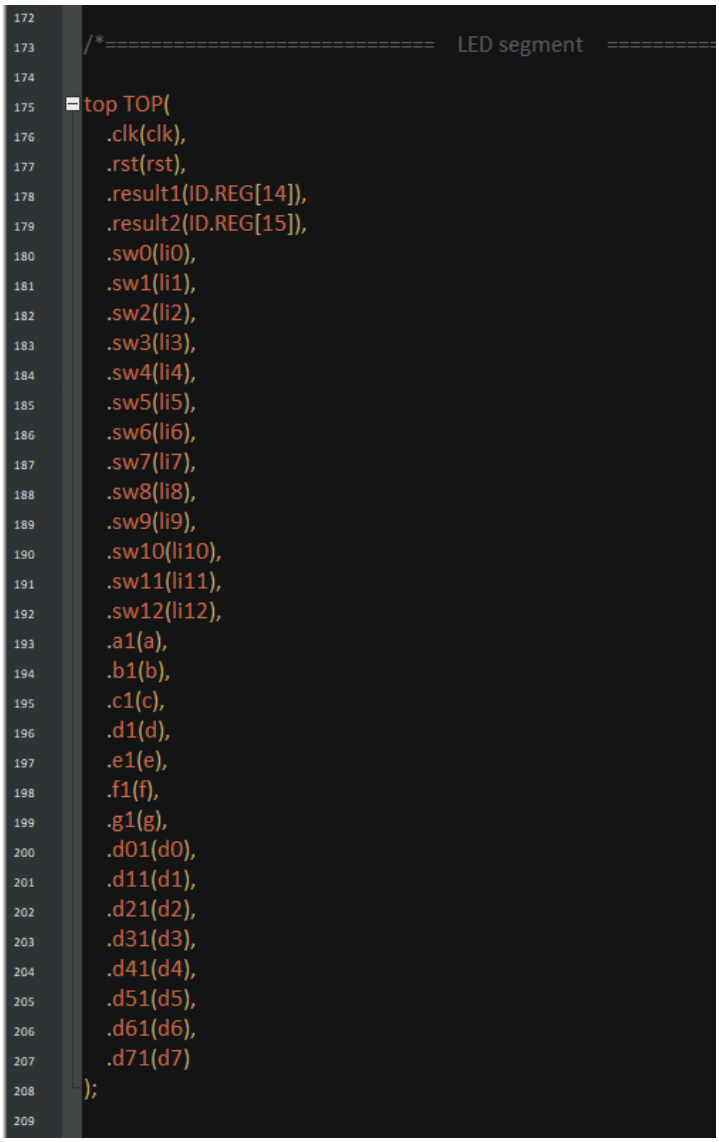


```
/*============================ INSTRUCTION_DECODE ============================*/

INSTRUCTION_DECODE ID(
    .clk(clk),
    .rst(rst),
    .PC(FD_PC),
    .IR(FD_IR),
    .MW_MemtoReg(MW_MemtoReg),
    .MW_RegWrite(MW_RegWrite),
    .MW_RD(MW_RD),
    .MDR(MDR),
    .MW_ALUout(MW_ALUout),
    .sw0(li0),
    .sw1(li1),
    .sw2(li2),
    .sw3(li3),
    .sw4(li4),
    .sw5(li5),
    .sw6(li6),
    .sw7(li7),
    .sw8(li8),
    .sw9(li9),
    .sw10(li10),
    .sw11(li11),
    .sw12(li12),

    .MemtoReg(DX_MemtoReg),
    .RegWrite(DX_RegWrite),
    .MemRead(DX_MemRead),
    .MemWrite(DX_MemWrite),
    .branch(DX_branch),
    .jump(DX_jump),
    .ALUctr(ALUctr),
    .JT(DX_JT),
    .DX_PC(DX_PC),
    .NPC(DX_NPC),
    .A(A),
    .B(B),
    .imm(imm),
    .RD(DX_RD),
    .MD(DX_MD)
);
```

IF.v 的程式運算結果會存在 ID.REG[14]、ID.REG[15]，並將他們存入 result1、result2 中



```
/*============================  LED segment  ==========

top TOP(
    .clk(clk),
    .rst(rst),
    .result1(ID.REG[14]),
    .result2(ID.REG[15]),
    .sw0(li0),
    .sw1(li1),
    .sw2(li2),
    .sw3(li3),
    .sw4(li4),
    .sw5(li5),
    .sw6(li6),
    .sw7(li7),
    .sw8(li8),
    .sw9(li9),
    .sw10(li10),
    .sw11(li11),
    .sw12(li12),
    .a1(a),
    .b1(b),
    .c1(c),
    .d1(d),
    .e1(e),
    .f1(f),
    .g1(g),
    .d01(d0),
    .d11(d1),
    .d21(d2),
    .d31(d3),
    .d41(d4),
    .d51(d5),
    .d61(d6),
    .d71(d7)
);
```

TOP.v 中，在 rst==1 的時候，顯示 sw12...sw0 的結果，讓指撥的結果直接顯示出來，並且 rst==0 的時候，開始做運算。

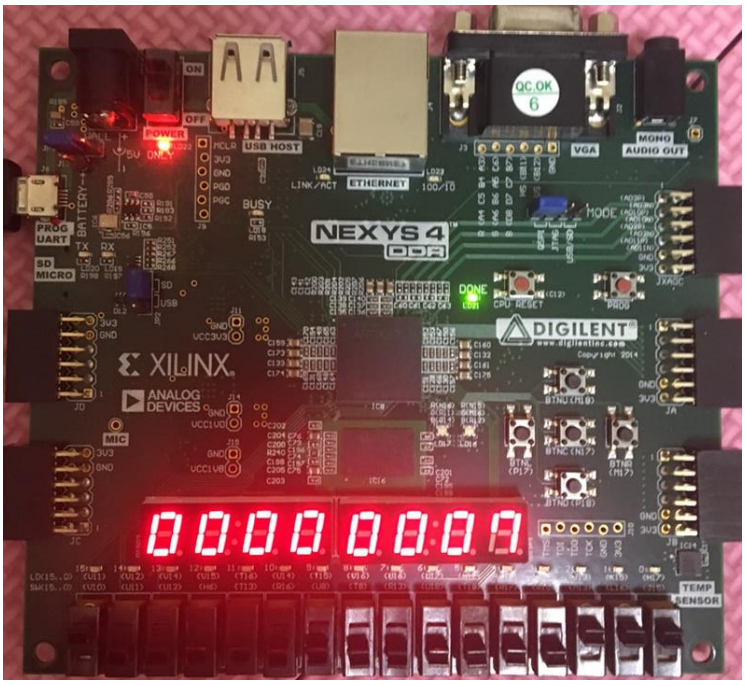
```
        answer_number1 <= 13'd0;
        answer_number2 <= {sw12, sw11, sw10, sw9, sw8, sw7, sw6, sw5, sw4, sw3, sw2, sw1, sw0};
    end
    //second_number <= {sw5,sw4,sw3,sw2,sw1,sw0};
    //first_number <= {sw11,sw10,sw9,sw8,sw7,sw6};
    else begin
        answer_number1 <= result2[12:0];
        answer_number2 <= result1[12:0];
    end
end

//8顆(d0~d7)7-segment(a~g)顯示 dp為右下角的.
assign {d71,d61,d51,d41,d31,d21,d11,d01} = scan; //亮哪一顆LED
//assign dp = ((state==1) || (state==3)) ? 0 : 1;  //1,3 light_on
always@(posedge clk) begin
  counter <=(counter<=100000) ? (counter +1) : 0;
  state <= (counter==100000) ? (state + 1) : state;

    case(state)
        0:begin
            //seg_number <= first_number/10;//6個switch值最多為63,63/10=6,顯示在左邊
            seg_number <= answer_number1/1000;
            scan <= 8'b0111_1111;
        end
        1:begin
            //seg_number <= first_number%10;//63%10=3,顯示在右邊
            seg_number <= (answer_number1/100)%10;
            scan <= 8'b1011_1111;
        end
        2:begin
            //seg_number <= second_number/10;
            seg_number <= (answer_number1/10)%10;
            scan <= 8'b1101_1111;
        end
```

如何控制顯示器

1. 我把 sw15 當作 rst，往上可以調數字，往下顯示運算結果
2. 往上時的數字會顯示在螢幕上。



3. 往下時的數字，左邊的是大於 input 的最接近質數，右邊的是小於 input 的最接近質數。