实现数据处理指令的 CPU 设计实验报告

2021 年 5 月 24 日　　　　　　　　　　　　成绩：

姓名	周俊佐	学号	19071337	班级	19185312
姓名	颜伟鹏	学号	19071334	班级	19185312
姓名	高炜哲	学号	18081811	班级	18181411
专业	计算机科学与技术		课程名称	计算机组成原理课程设计	
任课老师	章复嘉	指导老师	章复嘉	机位号	95
实验序号	8	实验名称	实现数据处理指令的 CPU 设计实验		

一、实验目的和实验要求

1.掌握 ARM V7-A 数据处理指令三种格式的数据通路设计，实现数据处理指令的功能，掌握指令流和数据流的控制方法；

2.在前面实验的基础上，用有限状态机编程方式编写一个 CPU 模块，进行模块连接，能够实现指定的 8 条数据处理指令。

3.编写一个实验验证的顶层模块，用实验测试程序进行整机测试，完成仿真调试和板级调试；

4.撰写实验报告。

二、实验程序源代码

ALU.V

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
module ALU(
    // INPUT
    input [31:0] A,
    input [3:0] ALU_OP,
    input CF, // from CSPR
    input VF, // from CSPR
    // FROM BARRELSHIFTER
    input  [31:0] B,
    input  Shift_Carry_Out,
    // OUTPUT
    output reg [31:0] F, // result
    output reg [3:0] NZCV // push into CSPR
);
    // C32 FLAG
    reg C32;

    // ALU_CF:
    // ADD: C32 = 1 -> C = 1
    // SUB: C32 = 1 -> C = 0
    // ALU_VF: C32 ^ C31

    always@(*) begin
    case(ALU_OP)
         4'b0000:begin
            F <= A & B;
            // C = NZCV[1]
            NZCV[1] <= Shift_Carry_Out;
            // V = NZCV[0]
            NZCV[0] <= VF;end
         4'b0001:begin
            F = A ^ B;
            // C = NZCV[1]
            NZCV[1] <= Shift_Carry_Out;
            // V = NZCV[0]
            NZCV[0] <= VF;end
         4'b0010:begin
            {C32, F} <= A - B;
            // C = NZCV[1]
            NZCV[1] <= ~C32;
            // V = NZCV[0]
            NZCV[0] <= C32 ^ F[31];end
        4'b0011:begin
            {C32, F} <= B - A;
            // C = NZCV[1]
            NZCV[1] <= ~C32;
            // V = NZCV[0]
            NZCV[0] <= C32 ^ F[31];end
        4'b0100:begin
            {C32, F} <= A + B;
            // C = NZCV[1]
            NZCV[1] <= C32;
            // V = NZCV[0]
            NZCV[0] <= C32 ^ F[31];end
        4'b0101:begin
            {C32, F} <= A + B + CF;
            // C = NZCV[1]
            NZCV[1] <= C32;
            // V = NZCV[0]
            NZCV[0] <= C32 ^ F[31];end
        4'b0110:begin
            {C32, F} <= A - B + CF - 1;
            // C = NZCV[1]
            NZCV[1] <= ~C32;
            // V = NZCV[0]
            NZCV[0] <= C32 ^ F[31];end
        4'b0111:begin
            F <= B - A + CF - 1;
            // C = NZCV[1]
            NZCV[1] <= ~C32;
            // V = NZCV[0]
            NZCV[0] <= C32 ^ F[31];end
        4'b1000:begin
            F <= A;
            // C = NZCV[1]
            NZCV[1] <= Shift_Carry_Out;
            // V = NZCV[0]
            NZCV[0] <= VF;end
        4'b1010:begin
            F <= A - B + 4;
            // C = NZCV[1]
            NZCV[1] <= ~C32;
            // V = NZCV[0]
            NZCV[0] <= C32 ^ F[31];end
```

```
         4'b1100:begin
            F <= A|B;
            // C = NZCV[1]
            NZCV[1] <= Shift_Carry_Out;
            // V = NZCV[0]
            NZCV[0] <= VF;end
         4'b1101:begin
            F <= B;
            // C = NZCV[1]
            NZCV[1] <= Shift_Carry_Out;
            // V = NZCV[0]
            NZCV[0] <= VF;end
         4'b1110:begin
            F <= A & (~B);
            // C = NZCV[1]
            NZCV[1] <= Shift_Carry_Out;
            // V = NZCV[0]
            NZCV[0] <= VF;end
         4'b1111:begin
            F <= ~B;
            // C = NZCV[1]
            NZCV[1] <= Shift_Carry_Out;
            // V = NZCV[0]
            NZCV[0] <= VF;end
     endcase
     // N = NZCV[3]
     NZCV[3] <= F[31];
     // Z = NZCV[2]
     NZCV[2] <= (F == 0) ? 1 : 0;
     end

endmodule
```

ALU_barrelShifter.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
module ALU_barrelShifter(
    // input from shifter
    input [31:0] Shift_Data,
    input [31:0] Shift_Num,
    input [3:0] SHIFT_OP,
    // input from alu
    input [31:0] A,
    input [3:0] ALU_OP,
    input CF,
    input VF,
    // output
    output [31:0] F,
    output [3:0] NZCV
);
    wire [31:0] Shift_Out;
    wire Shift_Carry_Out;

    barrelShifter Shifter_Instance(
    .SHIFT_OP(SHIFT_OP),
    .Shift_Data(Shift_Data),
    .Shift_Num(Shift_Num),
    .Carry_flag(CF),
    .Shift_Out(Shift_Out),
    .Shift_Carry_Out(Shift_Carry_Out)
    );

    ALU ALU_Instance(
    .ALU_OP(ALU_OP),
    .A(A),
    .B(Shift_Out),
    .Shift_Carry_Out(Shift_Carry_Out),
    .CF(CF),
    .VF(VF),
    .NZCV(NZCV),
    .F(F)
    );

endmodule
```

barrelShifter.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
module barrelShifter(
    // INPUT
    input [31:0] Shift_Data, // Shift Data
    input [7:0] Shift_Num, // Shift bits
    // SHIFT_OP[2:1] shift function
    // SHIFT_OP[0] decide shift bits with Shift_Num
    input [2:0] SHIFT_OP, //Shift OP
```

```verilog
    input Carry_flag,
    // OUTPUT
    output reg [31:0] Shift_Out,
    output reg Shift_Carry_Out
);

    always@(*) begin
    case(SHIFT_OP[2:1])
        2'b00: begin
            if (Shift_Num == 0) begin
                Shift_Out <= Shift_Data;
                Shift_Carry_Out <= 1'bx;
            end
            else if (Shift_Num >= 1 && Shift_Num <= 32) begin
                Shift_Out <= (Shift_Data << Shift_Num);
                Shift_Carry_Out <= Shift_Data[32-Shift_Num];
            end
            else begin
                Shift_Out <= 0;
                Shift_Carry_Out <= 0;
            end
        end

        2'b01: begin
            if (Shift_Num == 0 && SHIFT_OP[0] == 1) begin
                    Shift_Out <= Shift_Data;
                    Shift_Carry_Out <= 1'bx;
            end
            else if (Shift_Num == 0 && SHIFT_OP[0] == 0) begin
                Shift_Out <= 0;
                Shift_Carry_Out <= Shift_Data[31];
            end
            else if (Shift_Num >= 1 && Shift_Num <= 32) begin
                Shift_Out <= (Shift_Data >> Shift_Num);
                Shift_Carry_Out <= Shift_Data[Shift_Num-1];
            end
            else begin
                Shift_Out <= 0;
                Shift_Carry_Out <= 0;
            end
        end

        2'b10: begin
            if (Shift_Num == 0 && SHIFT_OP[0] == 1) begin
```

```
                Shift_Out <= Shift_Data;
                Shift_Carry_Out <= 1'bx;
            end
            if (Shift_Num == 0 && SHIFT_OP[0] == 0) begin
                Shift_Out <={32{Shift_Data[31]}};
                Shift_Carry_Out <= Shift_Data[31];
            end
            else if (Shift_Num >= 1 && Shift_Num <= 31) begin
                Shift_Out <= ({{32{Shift_Data[31]}},Shift_Data}>>Shift_Num);
                Shift_Carry_Out <= Shift_Data[Shift_Num-1];
            end
            else begin
                Shift_Out <= {32{Shift_Data[31]}};
                Shift_Carry_Out <= Shift_Data[31];
            end
        end

        2'b11: begin
            if (Shift_Num == 0 && SHIFT_OP[0] == 1) begin
                    Shift_Out <= Shift_Data;
                    Shift_Carry_Out <= 1'bx;
            end
            else if (Shift_Num == 0 && SHIFT_OP[0] == 0) begin
                    Shift_Out <= {Carry_flag, Shift_Data[31:1]};
                    Shift_Carry_Out <= Shift_Data[0];
            end
            else if (Shift_Num >= 1 && Shift_Num <= 32) begin
                Shift_Out <= ({Shift_Data,Shift_Data}>>Shift_Num);
                Shift_Carry_Out <= Shift_Data[Shift_Num-1];
            end
            else begin
                Shift_Out <= ({{32{Shift_Data}},Shift_Data}>>Shift_Num[4:0]);
                Shift_Carry_Out <= Shift_Data[Shift_Num[4:0]-1];
            end
        end
    endcase
    end
```

Inst_Mod.v

```
`timescale 1ns / 1ps

module Inst_Mod(
    // input
```

```verilog
    input clk,
    input Rst,
    input Write_PC,
    input Write_IR,
    input [3:0] NZCV, // from CSPR[32:28]
    // output
    output flag,
    output [7:2] Inst_Addr,
    output  [31:28] cond,
    output [27:0] IR
);

wire [31:0] PC;
wire [31:0] Inst;
wire [31:0] IR_complete;

assign Inst_Addr = PC[7:2];
assign cond = IR_complete[31:28];
assign IR = IR_complete[27:0];

// PC
PC PC_Instance(
    // input
    .clk(clk),
    .Rst(Rst),
    .Write_PC(Write_PC),
    // output
    .PC(PC)
);

// Inst_Rom
Inst_ROM Inst_ROM_Instance(
    // input
    .clka(clk),
    .addra(Inst_Addr),
    // output
    .douta(Inst)
);

// Inst_Reg
Inst_Reg Inst_Reg_Instance(
    // input
    .clk(clk),
    .Rst(Rst),
    .NZCV(NZCV),
    .Write_IR(Write_IR),
    .Inst(Inst),
    // output
    .flag(flag),
    .IR_complete(IR_complete)
);
endmodule
```

Inst_Reg.v

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

module Inst_Reg(
    // input
    input clk,
    input Rst,
    input [3:0] NZCV, // from CSPR
    input [31:0] Inst, // Instruction from Inst_Rom
    input Write_IR,
    //output
    output reg flag,
    output reg [31:0] IR_complete
    );



    assign VF = NZCV[0];
    assign CF = NZCV[1];
    assign ZF = NZCV[2];
    assign NF = NZCV[3];

    always@(negedge clk or posedge Rst) begin
        // Clear
        if(Rst) begin
            IR_complete <= 32'h00000000;
        end
        // Write
        else begin
            if(Write_IR) begin
                case(Inst[31:28])
                    4'b0000: begin //EQ
                        if(ZF == 1) begin IR_complete <= Inst;
                        flag <= 1;end
                        else flag <= 0;
                    end
                    4'b0001: begin //NE
                        if(ZF == 0) begin IR_complete <= Inst;
                        flag <= 1;end
                        else flag <= 0;
                    end
                    4'b0010: begin //CS
                        if(CF == 1) begin IR_complete <= Inst;
                        flag <= 1;end
                        else flag <= 0;
                    end
                    4'b0011: begin //CC
                        if(CF == 0) begin IR_complete <= Inst;
                        flag <= 1;end
                        else flag <= 0;
                    end
                    4'b0100: begin //MI
                        if(NF == 1) begin IR_complete <= Inst;
                        flag <= 1;end
                        else flag <= 0;
                    end
                    4'b0101: begin //PL
                        if(NF == 0) begin IR_complete <= Inst;
                        flag <= 1;end
                        else flag <= 0;
                    end
                    4'b0110: begin //VS
                        if(VF == 1) begin IR_complete <= Inst;
                        flag <= 1;end
                        else flag <= 0;
                    end
                    4'b0111: begin //VC
                        if(VF == 0) begin IR_complete <= Inst;
                        flag <= 1;end
                        else flag <= 0;
                    end
                    4'b1000: begin //HI
                        if(CF == 1 && ZF == 0) begin IR_complete <= Inst;
                        flag <= 1;end
                        else flag <= 0;
                    end
                    4'b1001: begin //LS
                        if(CF == 0 && ZF == 1) begin IR_complete <= Inst;
                        flag <= 1;end
                        else flag <= 0;
                    end
                    4'b1010: begin //GE
                        if(NF == VF) begin IR_complete <= Inst;
                        flag <= 1;end
                        else flag <= 0;
                    end
                    4'b1011: begin //LT
                        if(NF != VF) begin IR_complete <= Inst;
                        flag <= 1;end
                        else flag <= 0;
                    end
                    4'b1100: begin //GT
                        if(ZF == 0 && NF == VF) begin IR_complete <= Inst;
                        flag <= 1;end
                        else flag <= 0;
                    end
                    4'b1101: begin //LE
                        if(ZF == 1 && NF != VF) begin IR_complete <= Inst;
                        flag <= 1;end
                        else flag <= 0;
                    end
                    4'b1110: begin //AL
                        IR_complete <= Inst;
                        flag <= 1;
                    end
                    default: begin
                        flag <= 0;
                    end
                endcase
            end
        end
    end
```

RegFile.v

```
`timescale 1ns / 1ps

module RegFile(clk, Rst, Write_Reg, R_Addr_A, R_Addr_B, R_Addr_C, W_Addr, W_Data, R_Data_A, R_Data_B, R_Data_C);
```

```
    parameter ADDR = 4;
    parameter NUM  = 1<<ADDR;
    parameter SIZE = 32;

    input clk;
    input Rst;
    input Write_Reg;
    input [ADDR-1:0] R_Addr_A, R_Addr_B, R_Addr_C;
    input [ADDR-1:0] W_Addr;
    input [SIZE-1:0] W_Data;

    output [SIZE-1:0] R_Data_A, R_Data_B, R_Data_C;

    reg [SIZE-1:0] regfiles[0:NUM-1];
    integer i;

    initial
        for(i = 0;i<NUM;i = i+1) regfiles[i]<= 0;

    always@(negedge clk or posedge Rst) begin
        if (Rst)
            for(i = 0; i<NUM; i = i+1) regfiles[i]<= 0;
        else
            if (Write_Reg) regfiles[W_Addr] <= W_Data;
    end

    assign R_Data_A = regfiles[R_Addr_A];
    assign R_Data_B = regfiles[R_Addr_B];
    assign R_Data_C = regfiles[R_Addr_C];

endmodule
```

tb_top_CPU.v

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////////
module tb_top_CPU();

    wire [31:0] Inst,A,B,C,F;
    wire [3:0] NZCV;
    wire [5:0] Inst_Addr;
    wire LA,LB,LC,LF;
    wire Write_PC,Write_IR,Write_Reg,S;
    wire rm_imm_s;
    wire [1:0] rs_imm_s;
    wire [3:0] ALU_OP;
    wire [2:0] SHIFT_OP;
    reg clk;
    reg Rst;

    top_CPU top_CPU_Instance(
        .clk(clk),
        .Rst(Rst),
        .Inst(Inst),
        .A(A),
        .B(B),
        .C(C),
        .F(F),
        .NZCV(NZCV),
        .Inst_Addr(Inst_Addr),
        .Write_PC(Write_PC),
        .Write_IR(Write_IR),
        .Write_Reg(Write_Reg),
        .LA(LA),
        .LB(LB),
        .LC(LC),
        .LF(LF),
        .rm_imm_s(rm_imm_s),
        .rs_imm_s(rs_imm_s),
        .ALU_OP(ALU_OP),
        .SHIFT_OP(SHIFT_OP),
        .S(S)
    );
```

```
    initial begin
        clk=0;
        Rst=1;
        #40;
        Rst=0;
    end

    always begin
        #10
        clk=~clk;
    end
endmodule
```

top_CPU.v

```verilog
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
`timescale 1ns / 1ps

module top_CPU(
    input clk,
    input Rst,
    output [31:0] Inst,
    output reg [31:0] A,B,C,F,
    output reg [3:0] NZCV,
    output [5:0] Inst_Addr,
    output reg Write_PC,Write_IR,Write_Reg,S,
    output reg rm_imm_s,
    output reg [1:0] rs_imm_s,
    output reg [3:0] ALU_OP,
    output [2:0] SHIFT_OP,
    output reg LA,LB,LC,LF
);

    wire flag;
    wire[31:28] cond;
    wire [27:0] IR;
    Inst_Mod Inst_Mod_Instance(
        .clk(clk),
        .Rst(Rst),
        .Write_IR(Write_IR),
        .Write_PC(Write_PC),
        .NZCV(NZCV),
        .flag(flag),
        .Inst_Addr(Inst_Addr),
        .cond(cond),
        .IR(IR));

    assign Inst = {cond,IR};
```

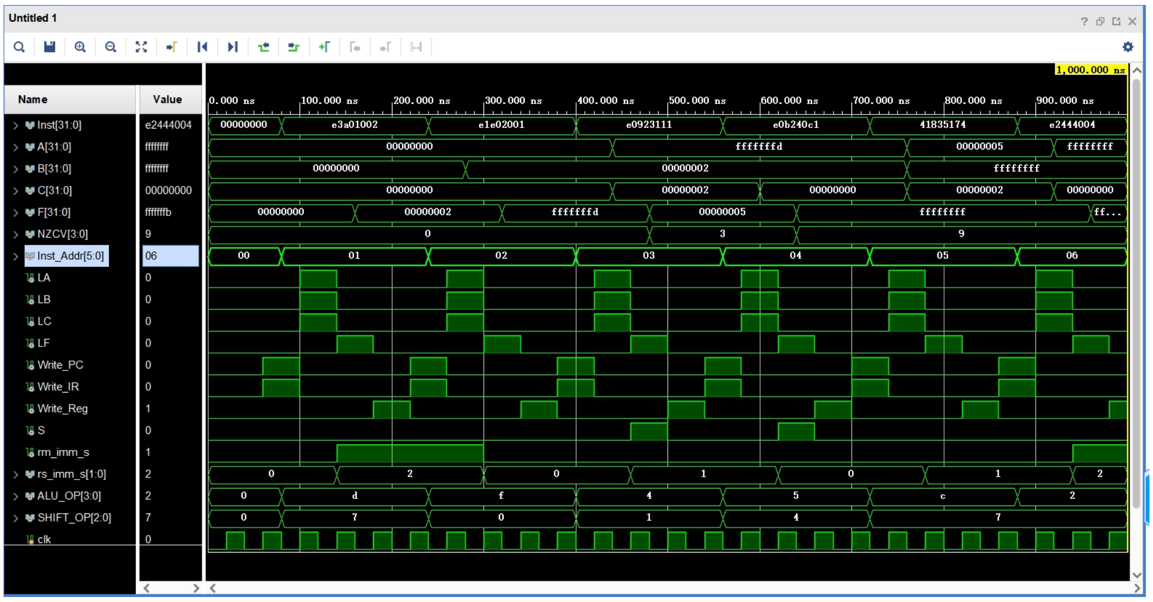
```verilog
    parameter DP0 = 2'd0;
    parameter DP1 = 2'd1;
    parameter DP2 = 2'd2;
    parameter Und = 2'd3;
    reg [1:0] DP;
    wire [3:0] OP,rn,rd,rs,rm;
    wire [4:0] imm5;
    wire [1:0] type;
```

```verilog
wire [11:0] imm12;

assign OP    = IR[24:21];
assign rn    = IR[19:16];
assign rd    = IR[15:12];
assign imm5  = IR[11:7];
assign rs    = IR[11:8];
assign type  = IR[6:5];
assign rm    = IR[3:0];
assign imm12 = IR[11:0];

always@(*)
begin
    if (&rd)
        DP = Und;
    else
    begin
        if (~|IR[27:25])
        begin
            if (!IR[4])
                DP = DP0;
            else if (!IR[7])
                DP = DP1;
            else
                DP = Und;
        end
        else if (IR[27:25] ^~ 3'b001)
            DP = DP2;
        else
            DP = Und;
    end
end

wire Und_Ins;
assign Und_Ins = DP == Und;

wire [31:0] R_Data_A,R_Data_B,R_Data_C;
RegFile RegFile_Instance(
    .clk(clk),
    .Rst(Rst),
    .Write_Reg(Write_Reg),
    .R_Addr_A(rn),
    .R_Addr_B(rm),
    .R_Addr_C(rs),
        .W_Addr(rd),
        .W_Data(F),
        .R_Data_A(R_Data_A),
        .R_Data_B(R_Data_B),
        .R_Data_C(R_Data_C));


    always@(negedge clk)
        if (LA) A <= R_Data_A;

    always@(negedge clk)
        if (LB) B <= R_Data_B;

    always@(negedge clk)
        if (LC) C <= R_Data_C;

    wire [7:0] Shift_Num;
    wire [31:0] Shift_Data;
    wire [31:0] F_New;
    wire [3:0] NZCV_New;
    assign Shift_Data = rm_imm_s?{{24{1'b0}},imm12[7:0]}:B;
    assign Shift_Num  = rs_imm_s[1]?{{3{1'b0}},imm12[11:8],1'b0}:(rs_imm_s[0]?C[7:0]:{{3{1'b0}},i
mm5});
    assign SHIFT_OP   = DP[1]?3'b111:{type,DP[0]};

    always@(*)
    begin
        if (OP[3] & !OP[2])
            ALU_OP = 4'b1000>>(4-OP[1:0]);
        else
            ALU_OP = OP;
    end

    ALU_barrelShifter ALU_barrelShifter_Instance(
        .SHIFT_OP(SHIFT_OP),
        .Shift_Data(Shift_Data),
        .Shift_Num(Shift_Num),
        .ALU_OP(ALU_OP),
        .A(A),
        .CF(NZCV[1]),
        .VF(NZCV[0]),
        .NZCV(NZCV_New),
        .F(F_New));
```

```
always@(negedge clk)
    if (S) NZCV <= NZCV_New;

always@(negedge clk)
    if (LF) F <= F_New;

localparam Idle = 3'd0;
localparam S0   = 3'd1;
localparam S1   = 3'd2;
localparam S2   = 3'd3;
localparam S3   = 3'd4;
reg [2:0] ST,Next_ST;

always@(posedge clk or posedge Rst)//状�?转移
begin
    if (Rst)
        ST <= Idle;
    else
        ST <= Next_ST;
end

always@(*)
begin
    Next_ST = Idle;
    case(ST)
        Idle:    Next_ST = S0;
        S0:      Next_ST = (flag & !Und_Ins)?S1: S0;
        S1:      Next_ST = S2;
        S2:      Next_ST = S3;
        S3:      Next_ST = S0;
        default: Next_ST = S0;
    endcase
end

always@(posedge clk or posedge Rst)
begin
    if (Rst)
    begin
        Write_PC  <= 1'b0;
        Write_IR  <= 1'b0;
        Write_Reg <= 1'b0;
        LA        <= 1'b0;
        LB        <= 1'b0;
        LC        <= 1'b0;
```

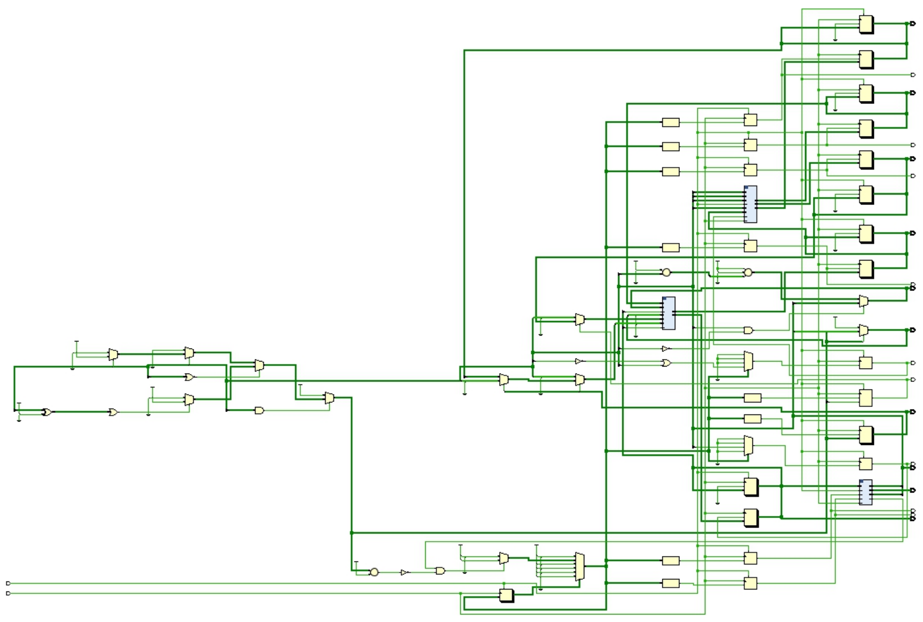
```
            LF         <= 1'b0;
            S          <= 1'b0;
            NZCV       <= 4'b0000;
            rm_imm_s   <= 1'b0;
            rs_imm_s   <= 2'b00;
            A          <= 0;
            B          <= 0;
            C          <= 0;
            F          <= 0;
        end
        else
        begin
            case(Next_ST)
                S0:begin
                    Write_PC   <= 1'b1;
                    Write_IR   <= 1'b1;
                    Write_Reg  <= 1'b0;
                    LA         <= 1'b0;
                    LB         <= 1'b0;
                    LC         <= 1'b0;
                    LF         <= 1'b0;
                    S          <= 1'b0;
                end
                S1:begin
                    Write_PC   <= 1'b0;
                    Write_IR   <= 1'b0;
                    Write_Reg  <= 1'b0;
                    LA         <= 1'b1;
                    LB         <= 1'b1;
                    LC         <= 1'b1;
                    LF         <= 1'b0;
                    S          <= 1'b0;
                end
                S2:begin
                    Write_PC   <= 1'b0;
                    Write_IR   <= 1'b0;
                    Write_Reg  <= 1'b0;
                    LA         <= 1'b0;
                    LB         <= 1'b0;
                    LC         <= 1'b0;
                    LF         <= 1'b1;
                    S          <= 1'b0;
                    rm_imm_s   <= DP[1];
                    rs_imm_s   <= DP;
                    S          <= IR[20];
                end
                S3:begin
                    Write_PC   <= 1'b0;
                    Write_IR   <= 1'b0;
                    Write_Reg <= !OP[3] | OP[2];
                    LA         <= 1'b0;
                    LB         <= 1'b0;
                    LC         <= 1'b0;
                    LF         <= 1'b0;
                    S          <= 1'b0;
                end
            endcase
        end

    end

endmodule
```

三、仿真文件和波形图（有就填，没有就不填）

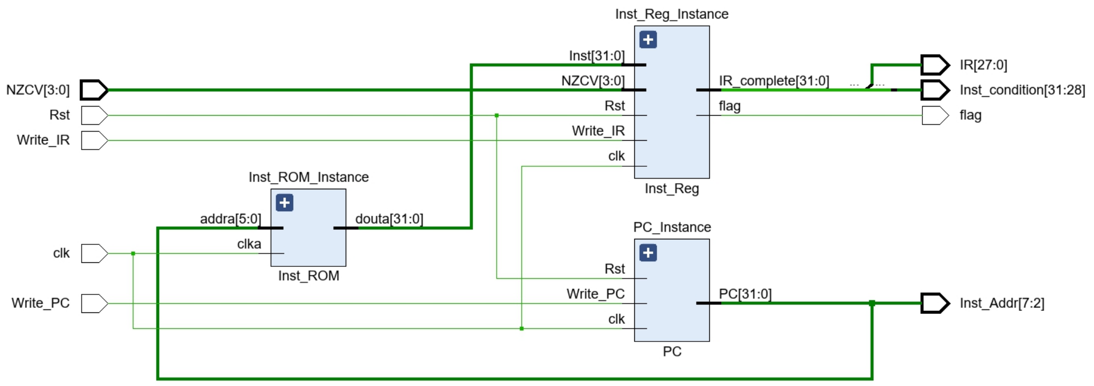


四、电路图（有就填，没有就不填）

总 CPU

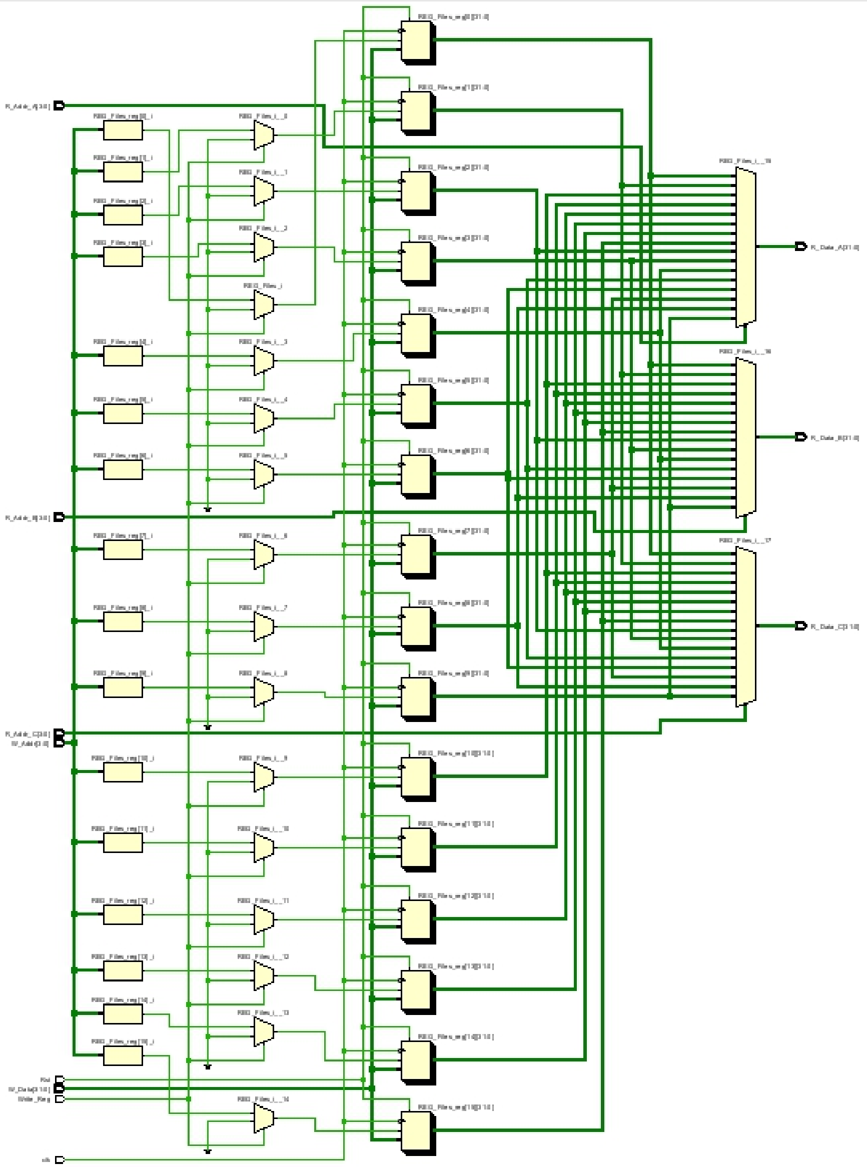


指令存取通路

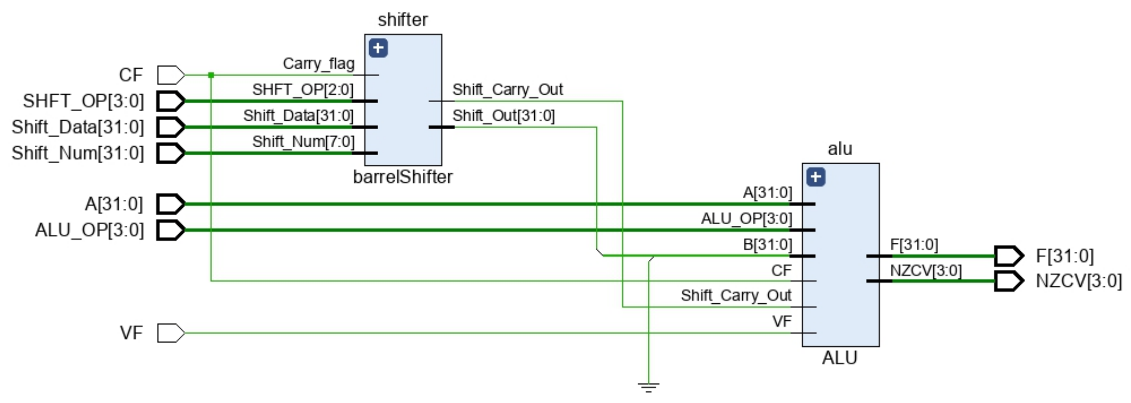


Inst_Reg_Instance
Inst[31:0]
NZCV[3:0]
Rst
Write_IR
clk
Inst_Reg
IR_complete[31:0]
flag
IR[27:0]
Inst_condition[31:28]
flag
NZCV[3:0]
Rst
Write_IR
Inst_ROM_Instance
addra[5:0]
clka
douta[31:0]
Inst_ROM
clk
Write_PC
PC_Instance
Rst
Write_PC
clk
PC
PC[31:0]
Inst_Addr[7:2]

寄存器堆



运算器



五、引脚配置（约束文件，有就填，没有就不填））

```
# .bit
set_property BITSTREAM.GENERAL.COMPRESS TRUE [current_design]

# Switch
set_property PULLDOWN true [get_ports sw]
set_property IOSTANDARD LVCMOS18 [get_ports sw]
set_property PACKAGE_PIN T3  [get_ports {sw[1]}]
set_property PACKAGE_PIN U3  [get_ports {sw[2]}]
set_property PACKAGE_PIN T4  [get_ports {sw[3]}]
set_property PACKAGE_PIN V3  [get_ports {sw[4]}]
set_property PACKAGE_PIN V4  [get_ports {sw[5]}]
set_property PACKAGE_PIN W4  [get_ports {sw[6]}]
set_property PACKAGE_PIN Y4  [get_ports {sw[7]}]
set_property PACKAGE_PIN Y6  [get_ports {sw[8]}]
set_property PACKAGE_PIN W7  [get_ports {sw[9]}]
set_property PACKAGE_PIN Y8  [get_ports {sw[10]}]
set_property PACKAGE_PIN Y7  [get_ports {sw[11]}]
set_property PACKAGE_PIN T1  [get_ports {sw[12]}]
set_property PACKAGE_PIN U1  [get_ports {sw[13]}]
set_property PACKAGE_PIN U2  [get_ports {sw[14]}]
set_property PACKAGE_PIN W1  [get_ports {sw[15]}]
set_property PACKAGE_PIN W2  [get_ports {sw[16]}]
set_property PACKAGE_PIN Y1  [get_ports {sw[17]}]
set_property PACKAGE_PIN AA1 [get_ports {sw[18]}]
set_property PACKAGE_PIN V2  [get_ports {sw[19]}]
set_property PACKAGE_PIN Y2  [get_ports {sw[20]}]
set_property PACKAGE_PIN AB1 [get_ports {sw[21]}]
set_property PACKAGE_PIN AB2 [get_ports {sw[22]}]
set_property PACKAGE_PIN AB3 [get_ports {sw[23]}]
set_property PACKAGE_PIN AB5 [get_ports {sw[24]}]
set_property PACKAGE_PIN AA6 [get_ports {sw[25]}]
set_property PACKAGE_PIN R2  [get_ports {sw[26]}]
set_property PACKAGE_PIN R3  [get_ports {sw[27]}]
set_property PACKAGE_PIN T6  [get_ports {sw[28]}]
set_property PACKAGE_PIN R6  [get_ports {sw[29]}]
set_property PACKAGE_PIN U7  [get_ports {sw[30]}]
set_property PACKAGE_PIN AB7 [get_ports {sw[31]}]
set_property PACKAGE_PIN AB8 [get_ports {sw[32]}]
```

```
# Switch Buttonť
set_property IOSTANDARD LVCMOS18 [get_ports swb]
set_property PACKAGE_PIN R4  [get_ports {swb[1]}]
set_property PACKAGE_PIN AA4 [get_ports {swb[2]}]
set_property PACKAGE_PIN AB6 [get_ports {swb[3]}]
set_property PACKAGE_PIN T5  [get_ports {swb[4]}]
set_property PACKAGE_PIN V8  [get_ports {swb[5]}]
set_property PACKAGE_PIN AA8 [get_ports {swb[6]}]
```

```
# LED
set_property PULLDOWN true [get_ports led]
set_property IOSTANDARD LVCMOS18 [get_ports led]
set_property PACKAGE_PIN R1 [get_ports {led[1]}]
set_property PACKAGE_PIN P2 [get_ports {led[2]}]
set_property PACKAGE_PIN P1 [get_ports {led[3]}]
set_property PACKAGE_PIN N2 [get_ports {led[4]}]
set_property PACKAGE_PIN M1 [get_ports {led[5]}]
set_property PACKAGE_PIN M2 [get_ports {led[6]}]
set_property PACKAGE_PIN L1 [get_ports {led[7]}]
set_property PACKAGE_PIN J2 [get_ports {led[8]}]
set_property PACKAGE_PIN G1 [get_ports {led[9]}]
set_property PACKAGE_PIN E1 [get_ports {led[10]}]
set_property PACKAGE_PIN D2 [get_ports {led[11]}]
set_property PACKAGE_PIN A1 [get_ports {led[12]}]
set_property PACKAGE_PIN L3 [get_ports {led[13]}]
set_property PACKAGE_PIN G3 [get_ports {led[14]}]
set_property PACKAGE_PIN K4 [get_ports {led[15]}]
set_property PACKAGE_PIN G4 [get_ports {led[16]}]
set_property PACKAGE_PIN K1 [get_ports {led[17]}]
set_property PACKAGE_PIN J1 [get_ports {led[18]}]
set_property PACKAGE_PIN H2 [get_ports {led[19]}]
set_property PACKAGE_PIN G2 [get_ports {led[20]}]
set_property PACKAGE_PIN F1 [get_ports {led[21]}]
set_property PACKAGE_PIN E2 [get_ports {led[22]}]
set_property PACKAGE_PIN D1 [get_ports {led[23]}]
set_property PACKAGE_PIN B1 [get_ports {led[24]}]
set_property PACKAGE_PIN B2 [get_ports {led[25]}]
set_property PACKAGE_PIN N3 [get_ports {led[26]}]
set_property PACKAGE_PIN M3 [get_ports {led[27]}]
set_property PACKAGE_PIN K3 [get_ports {led[28]}]
set_property PACKAGE_PIN H3 [get_ports {led[29]}]
set_property PACKAGE_PIN N4 [get_ports {led[30]}]
set_property PACKAGE_PIN L4 [get_ports {led[31]}]
set_property PACKAGE_PIN J4 [get_ports {led[32]}]
```

```
set_property IOSTANDARD LVCMOS18 [get_ports seg]
set_property PACKAGE_PIN H19 [get_ports {seg[7]}]
set_property PACKAGE_PIN G20 [get_ports {seg[6]}]
set_property PACKAGE_PIN J22 [get_ports {seg[5]}]
set_property PACKAGE_PIN K22 [get_ports {seg[4]}]
set_property PACKAGE_PIN K21 [get_ports {seg[3]}]
set_property PACKAGE_PIN H20 [get_ports {seg[2]}]
set_property PACKAGE_PIN H22 [get_ports {seg[1]}]
set_property PACKAGE_PIN J21 [get_ports {seg[0]}]
set_property IOSTANDARD LVCMOS18 [get_ports which]
set_property PACKAGE_PIN N22 [get_ports {which[0]}]
set_property PACKAGE_PIN M21 [get_ports {which[1]}]
set_property PACKAGE_PIN M22 [get_ports {which[2]}]
set_property -dict {IOSTANDARD LVCMOS18 PACKAGE_PIN L21} [get_ports enable]
set_property -dict {IOSTANDARD LVCMOS18 PACKAGE_PIN H4} [get_ports clk]
```

```
# [Place 30-574] Poor placement for routing between an IO pin and BUFG.If this
# sub optimal condition is acceptable for this design, you may use the
# CLOCK_DEDICATED_ROUTE constraint in the .xdc file to demote this message to a
# WARNING. However, the use of this override is highly discouraged.
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets clk_IBUF]
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets swb_IBUF[1]]
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets swb_IBUF[2]]
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets swb_IBUF[3]]
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets swb_IBUF[4]]
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets swb_IBUF[5]]
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets swb_IBUF[6]]
```

六、实验分工（需写清楚每个人负责的工作，以及个人贡献在小组总工作量中的占比）

总计：高炜哲负责 40%的实验，周俊佐和颜伟鹏各负责 30%的实验。

具体分工如下：

高炜哲：代码编写，代码调试，板卡调试

周俊佐：错误分析，测试数据，报告修改

颜伟鹏：实验报告，代码仿真测试。

七、实验收获（心得体会等，请以个人为单位分别写）

思考题：

(1) 请在本实验实现的 AND 和 EOR 指令基础上，分别 TST 和 TEQ 指令。

TST 即在实现 AND 的 ALU_OP 下，结果不写入目的寄存器，而是设置标志位，因此二进制指令码的 I[20]位为 1，即 S=1;

TEQ 即在实现 EOR 的 ALU_OP 下，结果不写入目的寄存器，而是设置标志位，因此二进制指令码的 I[20]位为 1，即 S=1;

(2) 请结合数据处理指令的 DP2 格式下的移位规则，阐述 DP2 格式的指令 AND r6,r3,#OxF000_0000 中，指令对应二进制码中的 imm12 为 0010_0000_1111 的原因。(#OxF000_0000 是移位以后的值)

DP2 采用循环右移的方式进行移位，且 immi12 的低 8 位为被移位数，高 4 位左移一次为移位次数;

被移位数：(0000_1111)B=（0F）H=（0000 000F）H
移位次数：(0010)B*2=(0100)B
移位方式：循环右移
结果（F000 0000)H，即为所求。

(3) 当目的寄存器(rd)==15 且 S==1 且 LR-4 →PC，OP==0010，立即数为#4 时，指令译码器应识别为 DP2 格式的 SUBS PC,LR,#4 指令，该指令的功能是实现异常返回。请在本实验实现的 CPU 结构和有限状态机基础上进行修改，实现 SUBS PC,LR,#4 指令。

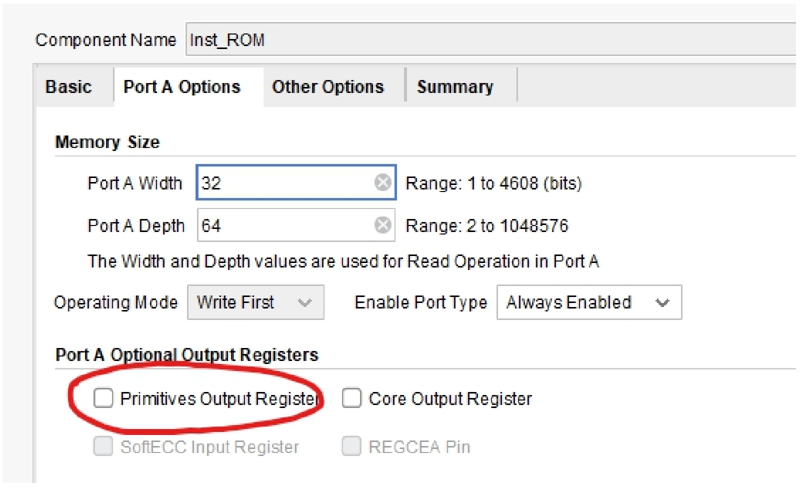
　　预取指令异常时，即 CPU 还没有自动更新 PC 值；出现预取指令异常后，要重新再执行一次这条指令，所以 PC 恢复的时候就需要 R14 减 4 并重新赋值给 PC。
因此当发生预取指令异常时，**进入异常状态假设为 S4，**在 S4 状态下，完成对 PC 的重新赋值后，返回先前的状态 S1，重新在 S1 状态下再次执行该条指令。
(4) 目的寄存器(rd)==15 且 S==1 且 LR→PC，OP==1101 时，指令译码器应识别为 DP0 格式 MOVS PC,LR 指令，该指令的功能是也是实现中断/异常返回。请在本实验实现的 CPU 结构和有限状态机基础上进行修改，实现 MOVS PC, LR 指令。

　　未定义指令异常时，CPU 还没有自动更新 PC 值；因为该指令未定义，所以返回时就不应该返回到这条未定义指令，而是返回到它的下一条指令，R14 中保存的刚好就是下一条指令的地址，所以就不用计算了，直接将 R14 赋值给 PC。
因此需要在状态转移时候进行改动，在状态 S1 时，若检测到 Und_Ins 则发生异常进入一个**新加入的状态假设为 S4**，在状态 S4 中完成上述指令的过程即跳过该未知指令；完成指令后重新返回 S0，执行下一条指令。

(5)说说你在实验中碰到了哪些问题，你是如何解决的?
1. 时钟错乱，读取指令时因某些原因会跳过部分指令，最终经过检查是 ROM 的配置问题。后来经过网络查询，发现勾选 primitives output register 后，将打开块 ROM 内部的位于输出数据总线之后的输出**流水线寄存器**，会使得块 ROM 输出的数据**延迟一拍**，导致一些异常。

感悟：
本次实验主要工作量是在 **1.**将之前编写的所有不见按照数据通路整合到一起 **2.**通过指令译码判断和状态转移形成一个有限状态机（其 CPU 本身就是一个复杂的有限状态机）在整合部件的过程中，主要增加的过程便是寄存器读出时有 LA LB LC3 个控制信号来控制数据的读出，以及 ALU 模块通过 LF 来控制运算结果的输出和 S 来控制是否修改 NZCV 状态寄存器。其次在状态转移的代码中，主要是通过状态的不同来改变 CPU 的所有多路选择信号和控制信号。其中

遇见的主要问题便是指令读取的时钟错乱，后发现是 ROM 配置问题，在上述的思考题(5)已有详细描述。配置图如下，并**不需要勾选** primitives output register:


Component Name: Inst_ROM

Basic | Port A Options | Other Options | Summary

Memory Size

Port A Width: 32 — Range: 1 to 4608 (bits)

Port A Depth: 64 — Range: 2 to 1048576

The Width and Depth values are used for Read Operation in Port A

Operating Mode: Write First — Enable Port Type: Always Enabled

Port A Optional Output Registers

☐ Primitives Output Register ☐ Core Output Register

☐ SoftECC Input Register ☐ REGCEA Pin

序号	指令	执行结果	标志位	结论
			NZCV	
1	MOV r1,#2	2→r1	0000	正常
2	MVN r2, r1, lsl #0	~r1→r2	0000	正常
序号	指令	执行结果	标志位	结论
3	ADDS r3, r2, r1, Isl r1	r2+(r1<<r1)→r3，更新标志位	0011	正常
4	ADCS r4, r2, r1, asr #1	r2+CF+(r1>>1)→r4，更新标志位	1001	正常
5	ORRMI r5, r3, r4, ror r1	if 上次运算结果为负 then r3 |({{32{r4[31]}}, r4}>>r1[4:0])→r5	1001	正常
6	SUB r4,#4	r4 -4→r4	1001	正常
7	AND r6, r3,#0x0F0000000	r3&0F000_0000H→r6	1001	正常
8	CMP r5, r6, lsl #0	r5- r6 ，更新标志位	1011	正常