# 课程报告

|  |  |  |  |
| --- | --- | --- | --- |
| 姓名 | 黄江晔 | 学号 | 22050201 |
| 专业 | 计算机科学与技术 | 班级 | 22052312 |
| 指导教师 | 冯建文 | 课程名称 | 计算机组成原理课程设计（甲） |
| 实验序号 | 3 | 实验名称 | 多功能ALU设计实验 |
| 实验时间 | 2024.4.24 | 实验地点 | 1教研楼225 |

# 成绩表

|  |  |  |
| --- | --- | --- |
| **考核项目** | **满分** | **得分** |
| **一、实验方案设计** | **20分** |  |
| **二、FPGA程序设计** | **40分** |  |
| **三、测试程序** | **10分** |  |
| **四、实验结果与分析** | **15分** |  |
| **五、思考与探索** | **10分** |  |
| **六、格式** | **5分** |  |
| **合计** | **100分** |  |

# 一、实验方案设计

## 实验目的与CPU目标指令集

（5分）

**实验目的：**

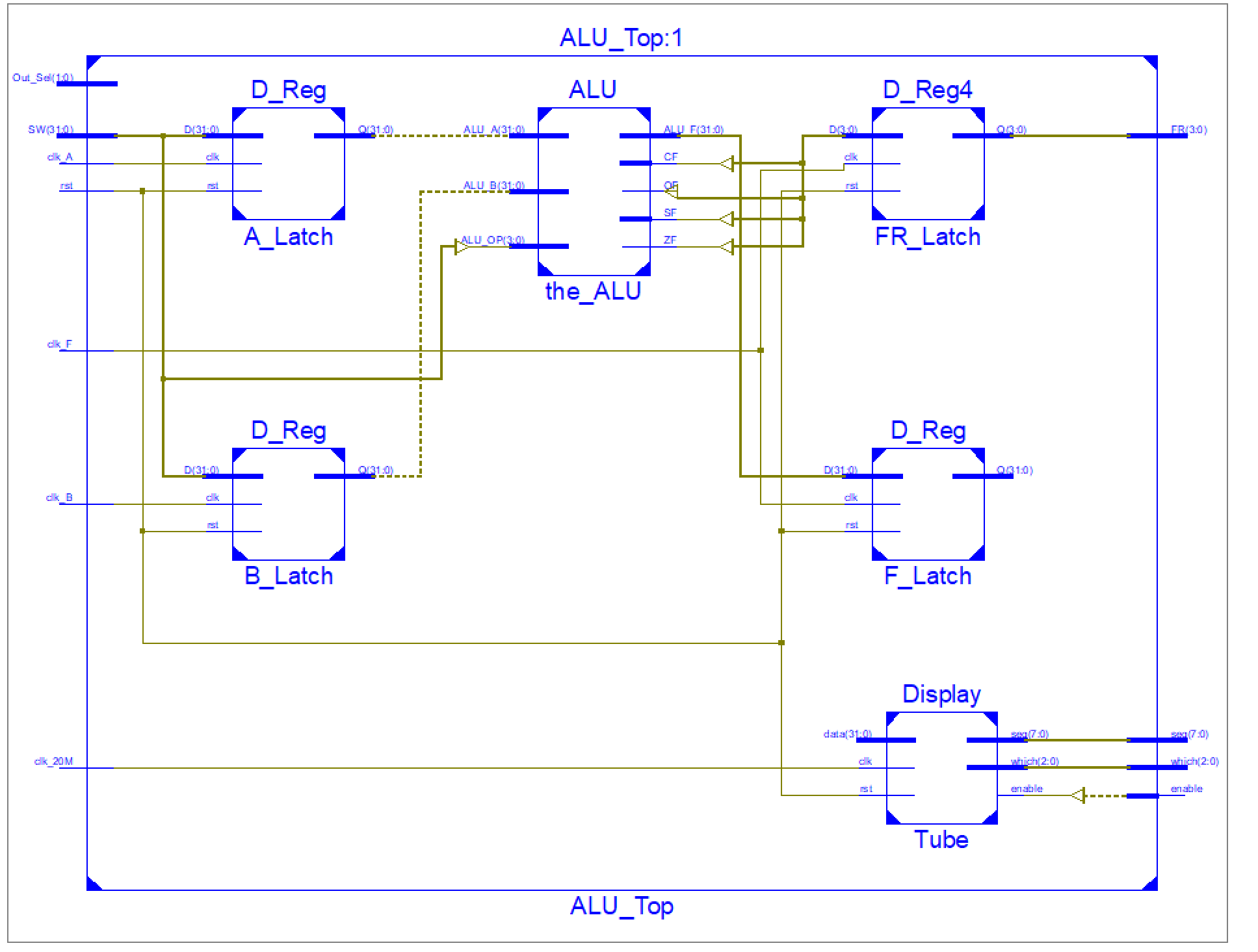
1. 学习多功能ALU的工作原理，掌握多功能ALU的设计方法；
2. 掌握暂存器的设计方法，及其与ALU的连接方法；
3. 掌握运用Verilog HDL语言进行组合逻辑电路与时序逻辑电路混合设计的方法；
4. 学会输入输出设备不足时的处理方法。

**CPU目标指令集：**

|  |  |  |
| --- | --- | --- |
| ALU\_OP | 运算 | 助记符 |
| 0000 | 加法：A+B | add |
| 0001 | 左移：A<<B | sll |
| 0010 | 有符号数比较小于置数：（A<B）?1:0 | slt |
| 0011 | 无符号数比较小于置数：（A<B）?1:0 | sltu |
| 0100 | 异或：A^B | xor |
| 0101 | 逻辑右移：A>>B,高位补0 | srl |
| 0110 | 按位或：A|B | or |
| 0111 | 按位与：A&B | and |
| 1000 | 减法：A-B | sub |
| 1101 | 算术右移：A>>B,高位补A[31] | sra |
| 其他 | 备用 | — |

## 模块构成与连接

（5分，请附图，并说明模型计算机的模块构成、各模块的功能，以及模块间的连接关系。也可附开发工具中显示的电路模块连接图，并加以说明。）



（1）**模块构成：**模型计算机由1个用于板级验证的顶层测试模块ALU\_Top，3个数据暂存器模块D\_Reg（A\_Latch，B\_Latch，F\_Latch），1个四位标志寄存器模块D\_Reg4，1个多功能ALU模块ALU，1个数码管显示模块Display构成。

（2）**各模块功能：**ALU模块实现满足RV32I指令集需求的、具有10种运算功能的32位ALU；D\_Reg模块实现将ALU操作状态的四个标志位情况暂时存储到寄存器中；3个暂存器将多功能ALU的输入数据和输出数据暂时存储到锁存器中；ALU\_Top模块是用于构造板级验证的顶层测试模块，来验证设计是否正确；Display模块使用数码管显示输入输出数据。

（3）**模块间的连接关系：**ALU\_Top顶层测试模块调用了D\_Reg A\_Latch模块和D\_Reg B\_Latch模块，分别将输入数据A和B输出到两个寄存器中，分别由时钟‘clk\_A’和‘clk\_B’控制；然后调用ALU模块，执行SW最低四位指定的操作，操作数是A和B，输出结果到ALU\_F，并设置标志位ZF,SF,CF,OF；调用D\_Reg F\_Latch模块存储来自ALU的结果ALU\_F，输出到F寄存器，时钟由‘clk\_F’控制；调用D\_Reg4 FR\_Latch模块，存储ALU的标志位并输出到FR；调用Display Tube显示模块负责将选定的数据，根据Out\_Sel选择显示A或B或F，显示到七段显示器。

## 板级验证方案（5分）

（5分，说明顶层测试模块的输入/输出信号，及对应使用的I/O设备；可附图说明）

**输入信号：**

（1）rst: 异步的复位信号，用于设置A、B、F、FR四个时序逻辑部件的初始状态。使用1个按键T5。

（2）clk\_A, clk\_B, clk\_F: 时钟信号，用于触发不同寄存器的更新。使用3个按键，分别为R4，AA4，AB6。

（3）SW: 32位数组，用于提供给ALU的输入数据，[3:0]是控制ALU操作的功能码。使用32个开关。

（4）clk\_20M: 20MHz的时钟信号。使用H4。

（5）Out\_Sel: 数码管的输出选择信号，用于选择要在数码管上显示的数据（01为A, 10为B,11为 F）。使用2个按键，V8和AA8。

**输出信号：**

（1）FR: 4位的标志寄存器，包括ALU操作后的零标志（ZF）、进位/借位标志（CF）、溢出标志（OF）、符号标志（SF）。接在LED指示灯上，分别是R1，P2，P1，N2。

（2）enable: 使能信号，用于控制数码管输出是否有效。在代码中设为1，即总是有效。

（3）which: 显示选择信号，用于选择数码管显示模块中的被激活的数码管。

（4）seg: 数码管显示信号，驱动七段显示灯具体应被点亮的段。

## FPGA管脚约束代码（5分）

（5分，与上面板级验证方案对应，给出引脚约束文件的内容，备注使用的具体设备）

**使用的具体设备：**

（1）rst、clk\_A、clk\_B、clk\_F使用的是四个按键，分别是T5，R4，AA4，AB6；

（2）SW使用的是32个开关；

（3）Out\_Sel使用的是剩下的两个按键，V8和AA8；

（4）FR使用四个LED灯显示；

（5）seg和which控制数码管显示。

**引脚约束文件具体如下：**

# PlanAhead Generated physical constraints

NET "SW[31]" LOC = T3;

# PlanAhead Generated IO constraints

NET "clk\_B" CLOCK\_DEDICATED\_ROUTE = FALSE;

NET "clk\_F" CLOCK\_DEDICATED\_ROUTE = FALSE;

NET "SW[31]" IOSTANDARD = LVCMOS18;

NET "SW[30]" IOSTANDARD = LVCMOS18;

NET "SW[29]" IOSTANDARD = LVCMOS18;

NET "SW[28]" IOSTANDARD = LVCMOS18;

NET "SW[27]" IOSTANDARD = LVCMOS18;

NET "SW[26]" IOSTANDARD = LVCMOS18;

NET "SW[25]" IOSTANDARD = LVCMOS18;

NET "SW[24]" IOSTANDARD = LVCMOS18;

NET "SW[23]" IOSTANDARD = LVCMOS18;

NET "SW[22]" IOSTANDARD = LVCMOS18;

NET "SW[21]" IOSTANDARD = LVCMOS18;

NET "SW[20]" IOSTANDARD = LVCMOS18;

NET "SW[19]" IOSTANDARD = LVCMOS18;

NET "SW[18]" IOSTANDARD = LVCMOS18;

NET "SW[17]" IOSTANDARD = LVCMOS18;

NET "SW[16]" IOSTANDARD = LVCMOS18;

NET "SW[15]" IOSTANDARD = LVCMOS18;

NET "SW[14]" IOSTANDARD = LVCMOS18;

NET "SW[13]" IOSTANDARD = LVCMOS18;

NET "SW[12]" IOSTANDARD = LVCMOS18;

NET "SW[11]" IOSTANDARD = LVCMOS18;

NET "SW[10]" IOSTANDARD = LVCMOS18;

NET "SW[9]" IOSTANDARD = LVCMOS18;

NET "SW[8]" IOSTANDARD = LVCMOS18;

NET "SW[7]" IOSTANDARD = LVCMOS18;

NET "SW[6]" IOSTANDARD = LVCMOS18;

NET "SW[5]" IOSTANDARD = LVCMOS18;

NET "SW[4]" IOSTANDARD = LVCMOS18;

NET "SW[3]" IOSTANDARD = LVCMOS18;

NET "SW[2]" IOSTANDARD = LVCMOS18;

NET "SW[1]" IOSTANDARD = LVCMOS18;

NET "SW[0]" IOSTANDARD = LVCMOS18;

# PlanAhead Generated physical constraints

NET "SW[30]" LOC = U3;

NET "SW[29]" LOC = T4;

NET "SW[28]" LOC = V3;

NET "SW[27]" LOC = V4;

NET "SW[26]" LOC = W4;

NET "SW[25]" LOC = Y4;

NET "SW[24]" LOC = Y6;

NET "SW[23]" LOC = W7;

NET "SW[22]" LOC = Y8;

NET "SW[21]" LOC = Y7;

NET "SW[20]" LOC = T1;

NET "SW[19]" LOC = U1;

NET "SW[18]" LOC = U2;

NET "SW[17]" LOC = W1;

NET "SW[16]" LOC = W2;

NET "SW[15]" LOC = Y1;

NET "SW[14]" LOC = AA1;

NET "SW[12]" LOC = Y2;

NET "SW[11]" LOC = AB1;

NET "SW[10]" LOC = AB2;

NET "SW[9]" LOC = AB3;

NET "SW[8]" LOC = AB5;

NET "SW[7]" LOC = AA6;

NET "SW[6]" LOC = R2;

NET "SW[5]" LOC = R3;

NET "SW[4]" LOC = T6;

NET "SW[3]" LOC = R6;

NET "SW[2]" LOC = U7;

NET "SW[1]" LOC = AB7;

NET "SW[0]" LOC = AB8;

# PlanAhead Generated IO constraints

NET "which[2]" IOSTANDARD = LVCMOS18;

NET "which[1]" IOSTANDARD = LVCMOS18;

NET "which[0]" IOSTANDARD = LVCMOS18;

NET "SW[31]" PULLDOWN;

NET "SW[30]" PULLDOWN;

NET "SW[29]" PULLDOWN;

NET "SW[28]" PULLDOWN;

NET "SW[27]" PULLDOWN;

NET "SW[26]" PULLDOWN;

NET "SW[25]" PULLDOWN;

NET "SW[24]" PULLDOWN;

NET "SW[23]" PULLDOWN;

NET "SW[22]" PULLDOWN;

NET "SW[21]" PULLDOWN;

NET "SW[20]" PULLDOWN;

NET "SW[19]" PULLDOWN;

NET "SW[18]" PULLDOWN;

NET "SW[17]" PULLDOWN;

NET "SW[16]" PULLDOWN;

NET "SW[15]" PULLDOWN;

NET "SW[14]" PULLDOWN;

NET "SW[13]" PULLDOWN;

NET "SW[12]" PULLDOWN;

NET "SW[11]" PULLDOWN;

NET "SW[10]" PULLDOWN;

NET "SW[9]" PULLDOWN;

NET "SW[8]" PULLDOWN;

NET "SW[7]" PULLDOWN;

NET "SW[6]" PULLDOWN;

NET "SW[5]" PULLDOWN;

NET "SW[4]" PULLDOWN;

NET "SW[3]" PULLDOWN;

NET "SW[2]" PULLDOWN;

NET "SW[1]" PULLDOWN;

NET "SW[0]" PULLDOWN;

# PlanAhead Generated physical constraints

NET "seg[7]" LOC = H19;

# PlanAhead Generated IO constraints

NET "seg[7]" IOSTANDARD = LVCMOS18;

NET "seg[6]" IOSTANDARD = LVCMOS18;

NET "seg[5]" IOSTANDARD = LVCMOS18;

NET "seg[4]" IOSTANDARD = LVCMOS18;

NET "seg[3]" IOSTANDARD = LVCMOS18;

NET "seg[2]" IOSTANDARD = LVCMOS18;

NET "seg[1]" IOSTANDARD = LVCMOS18;

NET "seg[0]" IOSTANDARD = LVCMOS18;

# PlanAhead Generated physical constraints

NET "seg[6]" LOC = G20;

NET "seg[5]" LOC = J22;

NET "seg[4]" LOC = K22;

NET "seg[3]" LOC = K21;

NET "seg[2]" LOC = H20;

NET "seg[1]" LOC = H22;

NET "seg[0]" LOC = J21;

# PlanAhead Generated IO constraints

NET "Out\_Sel[1]" IOSTANDARD = LVCMOS18;

NET "Out\_Sel[0]" IOSTANDARD = LVCMOS18;

NET "Out\_Sel[1]" PULLDOWN;

NET "Out\_Sel[0]" PULLDOWN;

# PlanAhead Generated physical constraints

NET "FR[3]" LOC = R1;

NET "FR[2]" LOC = P2;

NET "FR[1]" LOC = P1;

NET "FR[0]" LOC = N2;

# PlanAhead Generated IO constraints

NET "FR[3]" IOSTANDARD = LVCMOS18;

NET "FR[2]" IOSTANDARD = LVCMOS18;

NET "FR[1]" IOSTANDARD = LVCMOS18;

NET "FR[0]" IOSTANDARD = LVCMOS18;

# PlanAhead Generated physical constraints

NET "which[2]" LOC = M22;

NET "which[1]" LOC = M21;

NET "which[0]" LOC = N22;

NET "enable" LOC = L21;

# PlanAhead Generated IO constraints

NET "clk\_20M" IOSTANDARD = LVCMOS18;

NET "clk\_A" IOSTANDARD = LVCMOS18;

NET "clk\_B" IOSTANDARD = LVCMOS18;

NET "clk\_F" IOSTANDARD = LVCMOS18;

NET "enable" IOSTANDARD = LVCMOS18;

NET "rst" IOSTANDARD = LVCMOS18;

# PlanAhead Generated physical constraints

NET "clk\_20M" LOC = H4;

# PlanAhead Generated IO constraints

NET "clk\_20M" PULLDOWN;

# PlanAhead Generated physical constraints

NET "clk\_A" LOC = R4;

NET "clk\_B" LOC = AA4;

NET "clk\_F" LOC = AB6;

NET "rst" LOC = T5;

NET "SW[13]" LOC = V2;

NET "Out\_Sel[1]" LOC = V8;

NET "Out\_Sel[0]" LOC = AA8;

# PlanAhead Generated IO constraints

NET "clk\_A" PULLDOWN;

NET "clk\_B" PULLDOWN;

NET "clk\_F" PULLDOWN;

NET "rst" PULLDOWN;

# 二、FPGA程序设计

（请分模块给出代码（包含功能注释），并说明设计思路和功能。整体设计10分，各个模块30分。下面是一个样例，可以根据自己的具体设计增删改，也可以添加模块实验的仿真测试。）

## 顶层模块

**代码：**

`timescale 1ns / 1ps

`define X\_LEN 32

module ALU\_Top(

input rst,

input clk\_A,

input clk\_B,

input clk\_F,

input [`X\_LEN-1:0] SW,

output [3:0] FR,

input clk\_20M,

input [1:0] Out\_Sel,

output enable,

output [2:0] which,

output [7:0] seg

);

assign enable = 1'b1;

wire [3:0] Flags;

wire [`X\_LEN-1:0] A,B,F,ALU\_F;

wire ZF,SF,CF,OF;

D\_Reg A\_Latch(

.rst(rst),

.clk(clk\_A),

.D(SW),

.Q(A)

);

D\_Reg B\_Latch(

.rst(rst),

.clk(clk\_B),

.D(SW),

.Q(B)

);

ALU the\_ALU(

.ALU\_A(A),

.ALU\_B(B),

.ALU\_OP(SW[3:0]),//由最右边的4位开关提供运算功能

.ALU\_F(ALU\_F),

.ZF(ZF),

.SF(SF),

.CF(CF),

.OF(OF)

);

assign Flags = {ZF,SF,CF,OF};

D\_Reg F\_Latch(

.rst(rst),

.clk(clk\_F),

.D(ALU\_F),

.Q(F)

);

D\_Reg4 FR\_Latch(

.rst(rst),

.clk(clk\_F),

.D(Flags),

.Q(FR)

);

reg [31:0] Data;

Display Tube(

.rst(rst),

.clk(clk\_20M),

.data(Data),

.enable(enable),

.which(which),

.seg(seg)

);

always @(\*)

begin

case(Out\_Sel)

2'b01:Data=A;

2'b10:Data=B;

2'b11:Data=F;

default:Data=32'hE0E0\_E0E0;

endcase

end

endmodule

**设计思路：**

定义了ALU\_Top顶层模块，思路是调用D\_Reg A\_Latch模块和D\_Reg B\_Latch模块，分别将输入数据A和B输出到两个寄存器中，分别由时钟‘clk\_A’和‘clk\_B’控制；然后调用ALU模块，执行SW最低四位指定的操作，操作数是A和B，输出结果到ALU\_F，并设置标志位ZF,SF,CF,OF；调用D\_Reg F\_Latch模块存储来自ALU的结果ALU\_F，输出到F寄存器，时钟由‘clk\_F’控制；调用D\_Reg4 FR\_Latch模块，存储ALU的标志位并输出到FR；调用Display Tube显示模块负责将选定的数据，根据Out\_Sel选择显示A或B或F，显示到七段显示器。

**功能：**

将各子模块组合起来构造板级验证的顶层测试模块，来验证设计是否正确。

## CPU（整机）模块

（能实现目标指令集的CPU及整机模块）

**代码**：

`timescale 1ns / 1ps

module ALU(

input [31:0]ALU\_A,

input [31:0]ALU\_B,

input [3:0]ALU\_OP,

output reg [31:0]ALU\_F,

output ZF,

output SF,

output CF,

output OF

);

reg C32;

always@(\*)begin

C32=0;

case(ALU\_OP)

4'b0000:ALU\_F=ALU\_A+ALU\_B;//add加法

4'b0001:ALU\_F=ALU\_A<<ALU\_B;//sll左移

4'b0010:ALU\_F=$signed(ALU\_A)<$signed(ALU\_B)?1:0;//slt有符号数比较小于置数

4'b0011:ALU\_F=ALU\_A<ALU\_B?1:0;//sltu无符号数比较小于置数

4'b0100:ALU\_F=ALU\_A^ALU\_B;//xor异或

4'b0101:ALU\_F=ALU\_A>>ALU\_B;//srl逻辑右移

4'b0110:ALU\_F=ALU\_A|ALU\_B;//or按位或

4'b0111:ALU\_F=ALU\_A&ALU\_B;//and按位与

4'b1000:{C32,ALU\_F}=ALU\_A-ALU\_B;//sub减法

4'b1101:ALU\_F=$signed(ALU\_A)>>>ALU\_B;//sra算术右移

default:ALU\_F=32'b0;//其他备用

endcase

end

assign ZF = ~(|ALU\_F);

assign SF = ALU\_F[31];

assign CF = C32;

assign OF = C32 ^ ALU\_A[31] ^ ALU\_B[31] ^ ALU\_F[31];

endmodule

**设计思路：**

ALU模块输入是两个操作数A、B，四位操作码，输出是运算结果ALU\_F和四个标志位ZF，SF，CF，OF，通过case根据操作码选择对应的运算法则。

默认数据是无符号数，有符号数用$signed来指定，用补码表示。

**功能：**

ALU模块实现满足RV32I指令集需求的、具有10种运算功能的32位ALU。执行SW最低四位指定的操作，操作数是A和B，输出结果到ALU\_F，并设置标志位ZF,SF,CF,OF。

## 标志暂存器模块

**代码：**

`timescale 1ns / 1ps

module D\_Reg4(

input rst,

input clk,

input [3:0] D,

output reg[3:0] Q

);

always @(posedge rst or posedge clk)

begin

if(rst)

begin

Q<=4'b0;

end

else

begin

Q<=D;

end

end

endmodule

**设计思路：**

四位宽寄存器，在复位信号激活时，寄存器内容清零，而在时钟信号的上升沿且复位信号不激活的情况下，寄存器更新为新的标志位输入D。

**功能：**

D\_Reg模块实现将ALU操作状态的四个标志位情况（SF，OF，CF，ZF）暂时存储到寄存器中。

## 数据存储器模块

**代码**：

`timescale 1ns / 1ps

`define X\_LEN 32

module D\_Reg(

input rst,

input clk,

input [`X\_LEN-1:0] D,

output reg[`X\_LEN-1:0] Q

);

always @(posedge rst or posedge clk)

begin

if(rst)

Q<=`X\_LEN'b0;

else

Q<=D;

end

endmodule

**设计思路**：  
 用于存储输入和输出的32位数据。当复位信号激活时，寄存器内容被清零；当时钟信号的上升沿到达而复位信号未激活时，寄存器则加载新的数据输入D。

**功能：**

数据存储器模块将多功能ALU的输入数据和输出数据暂时存储到锁存器中。

## 数码管显示模块

**代码：**

`timescale 1ns / 1ps

// 8位 七段数码管扫描显示模块

module Display(

input clk, //时钟源20MHz

input rst, //复位信号

input [31:0] data, //32位待显示数据

output enable, //数码管显示使能，=1，某个（which指定）数码管点亮，=0，全灭

output reg [2:0] which, //片选编码（驱动哪一位数码管点亮）

output reg [7:0] seg // 段选信号（点亮哪些段，以显示字形）

);

reg [14:0] count = 0; // 分频扫描，从左至右循环驱动每一位数码管

always @(posedge rst or posedge clk)

begin

if(rst) count <= 0;

else count <= count + 1'b1;

end

always @(posedge rst or posedge count[14])

begin

if(rst) which <= 0;

else which <= which + 1'b1;

end

reg [3:0] digit; // 选择当前点亮的数码管对应显示的数字

always @\* case (which)

0: digit = data[31:28]; // 最高位数码管

1: digit = data[27:24];

2: digit = data[23:20];

3: digit = data[19:16];

4: digit = data[15:12];

5: digit = data[11:8];

6: digit = data[7:4];

7: digit = data[3:0]; // 最低位数码管

endcase

always @\* case (digit) // 根据当前要显示的数字，点亮数码管的对应段，显示字形

4'h0: seg = 8'b0000\_0011; // 除g、dp外全亮，显示数字0

4'h1: seg = 8'b1001\_1111; // 仅b、c亮，显示数字1

4'h2: seg = 8'b0010\_0101;

4'h3: seg = 8'b0000\_1101;

4'h4: seg = 8'b1001\_1001;

4'h5: seg = 8'b0100\_1001;

4'h6: seg = 8'b0100\_0001;

4'h7: seg = 8'b0001\_1111;

4'h8: seg = 8'b0000\_0001;

4'h9: seg = 8'b0000\_1001;

4'hA: seg = 8'b0001\_0001;

4'hB: seg = 8'b1100\_0001;

4'hC: seg = 8'b0110\_0011;

4'hD: seg = 8'b1000\_0101;

4'hE: seg = 8'b0110\_0001;

4'hF: seg = 8'b0111\_0001;

endcase

endmodule // Display

**设计思路：**

本模块Display设计为一个8位七段数码管扫描显示模块，用于将32位输入数据data映射到8个数码管上进行动态显示。该设计使用20MHz的时钟信号clk和复位信号rst来控制显示逻辑。通过一个15位计数器count，模块实现了从左至右的循环扫描，周期性地刷新每个数码管的显示内容。计数器每递增至最高位（第15位）时触发数码管的切换，确保每个数码管能均匀地获取足够的显示时间，以消除视觉闪烁。根据当前激活的数码管（由which寄存器指示），选择性地从32位数据中提取相应的4位并转换为相应的数码管字形显示。

**功能：**

Display模块主要功能是将32位输入数据通过内部逻辑映射到8个数码管上，实现动态显示。模块内部逻辑根据时钟信号和复位信号的状态动态更新显示数据。具体来说，模块使用一个15位计数器在每个时钟周期递增，当计数器的最高位发生变化时（即每32768个时钟周期），which寄存器的值递增，用于选择下一个要激活的数码管。每个数码管对应的显示数字是从32位数据中分段提取的，根据选定的数码管（which寄存器）提取对应的4位数字，并通过一系列的条件判断（case语句）转换为数码管的段选信号seg，从而控制数码管显示相应的数字或符号。

# 三、测试程序

## CPU测试程序

（8分，汇编语言程序、机器指令代码及对应主存地址、每条指令的预期执行结果）

## 存储器内容

（2分，指令存储器和数据存储器内容，可附coe文件内容）

# 四、实验结果与分析

（如果完成了板级验证，可以只写板级验证部分；具体按照任课教师要求。）

## 仿真实验结果与分析

### （1）仿真代码

（5分，只对模型机CPU模块仿真，不对顶层测试模块仿真，含仿真源代码、仿真验证方案）

### （2）仿真波形

（5分，仿真波形截图）

### （3）仿真波形与结果分析

（5分，对仿真波形进行分析）

## 板级实验结果与分析

### （1）板级实验操作说明

（5分，说明操作的顺序，应该如何观察、有何预期结果）

### （2）板级实验结果记录

（5分，以表格形式记录并展现，含汇编指令、PC值、IR值、计算结果F或者写入目的寄存器的值等运行结果）

### （3）实验结果分析与结论

（5分，分析实验结果，给出实验结论）

# 五、思考与探索

## 问题与解决方案

（3分，整个实验过程中发生了什么问题？你是如何解决的）

问题：在管脚配置中，32个开关用于32位的SW，Out\_Sel需要两个开关，而板卡开关数量不够。

解决：时钟和复位信号使用了四个按键，还剩下两个，用于Out\_Sel设置。

## 思考题

（7分，力所能及，除去上述“1.问题与解决方案”之外，尝试实践或回答教材上的思考与探索题目，至少完成1道）

# 六、实验心得体会、意见建议

（如果是小组合作，请给出具体分工、分别给出心得体会）