## 实验三 算术逻辑单元（ALU）的设计与实现

### 实验目的

1. 了解算术逻辑单元的功能。
2. 掌握算术逻辑单元的结构与实现。
3. 进一步熟练掌握Logisim 、Modelsim、Vivado软件和Verilog硬件描述语言的使用。

### 实验原理

1. ALU

算术逻辑单元（Arithmetic/Logic Unit , ALU）是现代计算机的核心部件之一。其内部由算术和逻辑操作部件组合而成，可以实现整数加、减等算术运算和与、或等逻辑运算。

一个典型的算术逻辑单元由两路N位输入、一路N位输出、一组功能选择信号和一些标志位（flag）组成。两路N位输入数据作为参与运算的两个操作数，输入到ALU中，通过改变功能选择信号，控制ALU对两操作数进行不同的算术或逻辑运算操作，并将N位的结果输出，与结果一同输出的还有运算产生的标志位，例如运算结果为零的标记信号Z（zero）与运算结果溢出的标记信号O（over）、进位标志C（CY）、符号位S（SF）等，如图2-1所示。

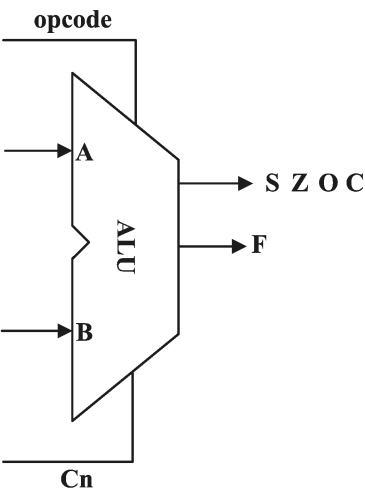


图2- 1ALU模块示意图

在本次实验中，我们把输入和输出的数据长度定为4位，数据输入命名为A、B，数据输出为F，功能控制信号输入为opcode，进位输入为Cn，只产生结果为零的zero标志位。

1. 功能控制

功能控制信号opcode的定义如表2-1所示。例如：opcode=0001，运算器实现加法运算。

表3- 1 ALU控制信号定义

|  |  |
| --- | --- |
| opcode | ALU运算 |
| 0000 | MOV |
| 0001 | ADD |
| 0010 | ADDC |
| 0011 | SUB |
| 0100 | SUBB |
| 0101 | AND |
| 0110 | OR |
| 0111 | NOT |
| 1000 | XOR |
| 1001 | INC |
| 1010 | DEC |

### 

### 实验内容

1. **基础实验**。用实验调试软件验证ALU的功能，并操作分析、记录结果。

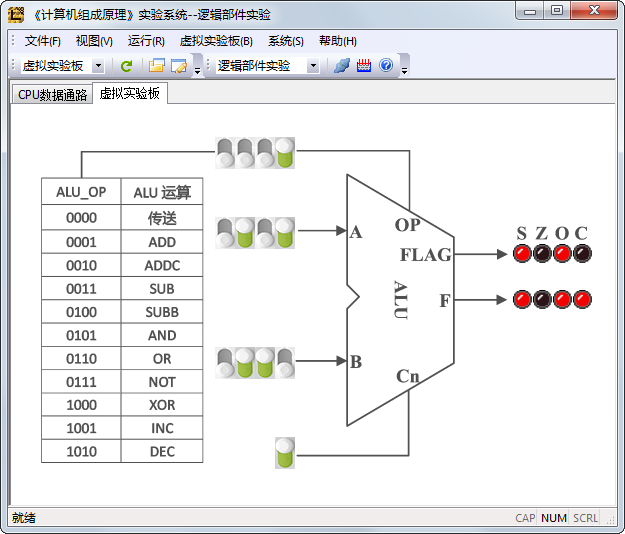


图2- 2 ALU虚拟实验示意图

#### （1）运算功能和控制信号

① 输入输出对于的开关指示灯分配如下：

输入信号 A:SW(4-7) B:SW(3-0) Cn:SW8 opcode:SW(12-9)

输出信号 F：LD(12-15) 标志位 S :LD(3) Z LD(2) O: LD(1) C: LD(0)

② 各种运算对应的控制信号及功能，如下表。

表3-2 ALU运算功能和控制信号定义

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 运算指令 | 运算功能 | 运算控制信号opcode | | | |
| M3 | M2 | M1 | M0 |
| MOV | F=A | 0 | 0 | 0 | 0 |
| ADD | F=A+B | 0 | 0 | 0 | 1 |
| SUB | F=A-B | 0 | 0 | 1 | 1 |
| INC | F=A+1 | 1 | 0 | 0 | 1 |
| DEC | F=A-1 | 1 | 0 | 1 | 0 |
| AND | F=A ∧ B | 0 | 1 | 0 | 1 |
| OR | F=A ∨ B | 0 | 1 | 1 | 0 |
| NOT | F= | 0 | 1 | 1 | 1 |
| XOR | F=A ⊕ B | 1 | 0 | 0 | 0 |
| ADDC | F=A+B+进位C0 | 0 | 0 | 1 | 0 |
| SUBB | F=A-B-借位C0 | 0 | 1 | 0 | 0 |

#### （2）加法运算结果的特征标志

设置opcode （M3~M0）为加法运算，按下表步骤操作，观察加法运算的结果，填入下表，并写出计算数和结果的真值。

|  | A | B | Cn | M3~M0 | F | FLAG | | | | 运算数和运算结果的真值 | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| S | Z | O | C | 视为无符号数 | 视为补码 |
| ① | 1000 | 0001 | — | 0001 | 1001 | 1 | 0 | 0 | 0 | 8＋1＝9 | (**﹣**8)＋1＝**﹣**7 |
| ② | 1101 | 1100 | — | 1001 | 1 | 0 | 0 | 1 | 13 + 12 =25 | ( -3) + (-4) = (-7) |
| ③ | 0100 | 0010 | — | 0110 | 0 | 0 | 0 | 0 | 4+ 2 = 6 | 4 + 2 = 6 |
| ④ | 0000 | 0000 | — | 0000 | 0 | 1 | 0 | 0 | 0 + 0 = 0 | 0 + 0 = 0 |
| ⑤ | 1111 | 0001 | — | 0000 | 0 | 1 | 0 | 1 | 15 + 1 = 16 | (-1) + 1 = 0 |
| ⑥ | 0011 | 0101 | — | 1000 | 1 | 0 | 1 | 0 | 3 + 5 = 8 | 3 + 5 = 8 |
| ⑦ | 1100 | 1011 | — | 0111 | 0 | 0 | 1 | 1 | 12 + 11 = 23 | (-4) + (-5) =(-9) |
| ⑧ | 1100 | 0101 | — | 0001 | 0 | 0 | 0 | 1 | 12 + 5 = 17 | (-4) + 5 = 1 |
| ⑨ | 0011 | 1011 | — | 1110 | 1 | 0 | 0 | 0 | 3 + 11 = 14 | 3 + (-5) = (-2) |
| ⑩ | 1000 | 1000 | — | 0000 | 0 | 1 | 1 | 1 | 8 + 8 = 16 | (-8) +(-8) = (-16) |

提示：为方便分析运算结果，可以事先列出负数的4位补码与真值的对应关系(上补下原)：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1000 | 1001 | 1010 | 1011 | 1100 | 1101 | 1110 | 1111 |
| -8无原码 | 1111(-7) | 1110(-6) | 1101(-5) | 1100(-4) | 1011(-3) | 1010(-2) | 1001(-1) |

实验现象分析：

（1）符号标志S就是运算结果的\_\_\_\_\_\_最高位\_\_\_\_\_（最高位／最低位）。

（2）零标志Z的生成和\_\_\_\_F\_\_\_\_（F／C／F及C）有关。

（3）溢出标志OF和进位标志\_\_\_没有\_\_\_\_\_（有／没有）直接的联系。

（4）对照标志位和真值，可以看出溢出标志OF是按照\_\_\_补码\_\_\_\_\_（无符号数／补码）的运算结果设置的；进位标志CF是按照\_\_\_\_无符号数\_\_\_\_（无符号数／补码）运算的结果设置的。也就是说，如果运算数是无符号数，运算结果是否溢出是由\_\_C\_\_\_（C／O）反映的；如果运算数是有符号补码数，运算结果是否溢出是由\_\_\_O\_\_（C／O）反映的。

（5）4位补码能表示数值的范围是\_\_\_\_\_(-8) ~ 7\_\_\_\_\_\_\_\_，4位无符号数能表示数值的范围是\_\_\_\_0 ~ 15\_\_\_\_\_\_\_。

（6）运算器电路是否“知道”运算数是有符号数还是无符号数？\_\_不知道\_\_\_\_\_\_\_

#### （3）减法运算

减法运算是转换为加法计算的。设置M3~M0为减法运算，注意观察B操作数、Cn和FLAG的C（进位）标志位。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | A | B | Cn | M3~M0 | F | C | 实验现象分析 |
| ① | 0010 | 0001 | — | 0011 | 0001 | 1 | \_\_有\_\_ (有/无)借位 |
| ② | 0001 | 0010 | — | 1111 | 0 | \_\_无\_\_(有/无)借位 |

实验现象分析：

①减法运算时， F=\_\_\_\_A - B\_\_\_\_\_\_\_\_\_\_\_

②C标志与减法运算有没有产生借位\_\_\_\_有\_\_\_\_（有/没有）关系，没有产生借位时，CF=\_\_0\_\_\_；减法运算产生借位时，CF=\_1\_\_\_\_。

#### （4）带借位的减法运算

设置M3~M0为带借位的减法运算，注意观察F和Ci的关系。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | A | B | Cn | M3~M0 | F | C | 实验现象分析 |
| ① | 0101 | 0011 | 1 | 0100 | 0010 | 1 | 带借位 |
| ② | 0101 | 0011 | 0 | 0001 | 1 | 不带借位 |

实验现象分析：

带借位得到正确结果，不带借位的结果错误

#### （5）加1和减1运算

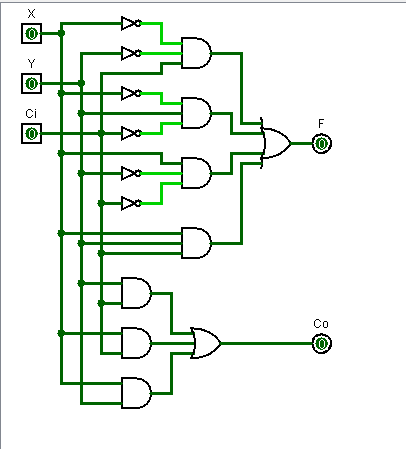
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | A | B | Cn | M3~M0 | F | FLAG  (S Z O C) |
| ① INC | 0010 | 0101 | — | 1001 | 0011 | 0 0 0 0 |
| ① INC | 0010 | 1010 | — | 0011 | 0 0 0 0 |
| ② DEC | 0010 | 1010 | — | 1010 | 0001 | 0 0 0 1 |
| ② DEC | 0010 | 0101 | — | 0001 | 0 0 0 1 |

实验现象分析：

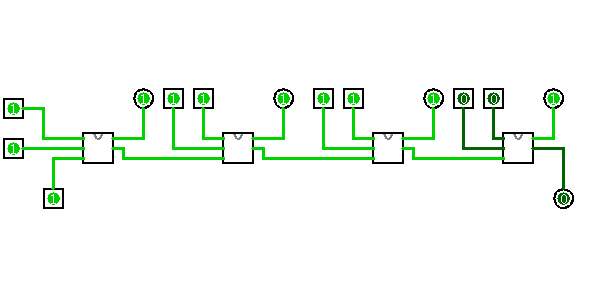
1. 加1运算时，B对结果\_\_无\_\_\_(有/没有)影响，F = \_\_\_A + 1\_\_\_\_\_。
2. 减1运算时，F = \_\_\_\_A - 1\_\_\_\_。

#### 提高实验。

1. Logisim实验
2. 利用Logisim设计一个1位全加器FA，并进行仿真测试。

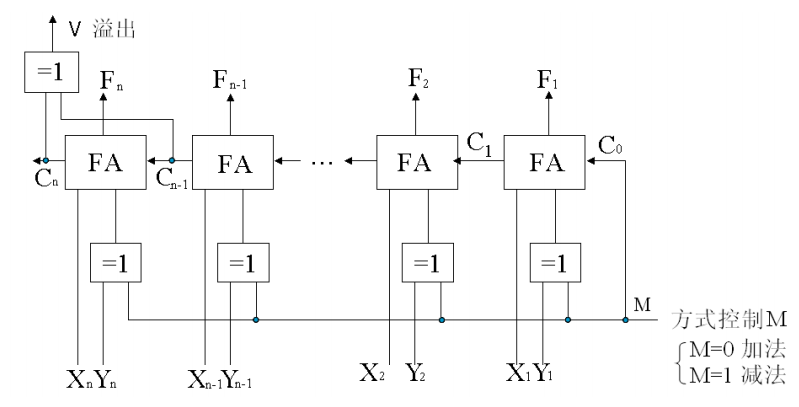


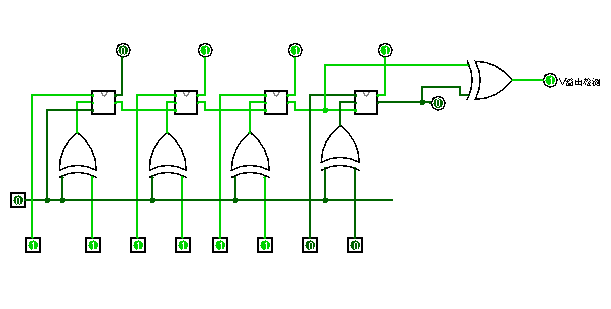
1. 利用（1）设计的1位全加器FA实现一个4位串行加法器，并进行仿真测试。



1. 利用（1）设计的1位全加器FA设计一个4位定点二进制补码加减法运算器，并进行仿真测试。PPT第三章P15 3.1.2

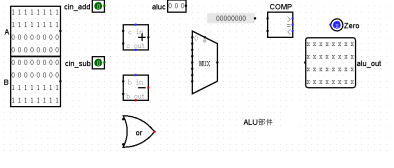
溢出详解：<https://blog.csdn.net/qq_42264877/article/details/103227523>



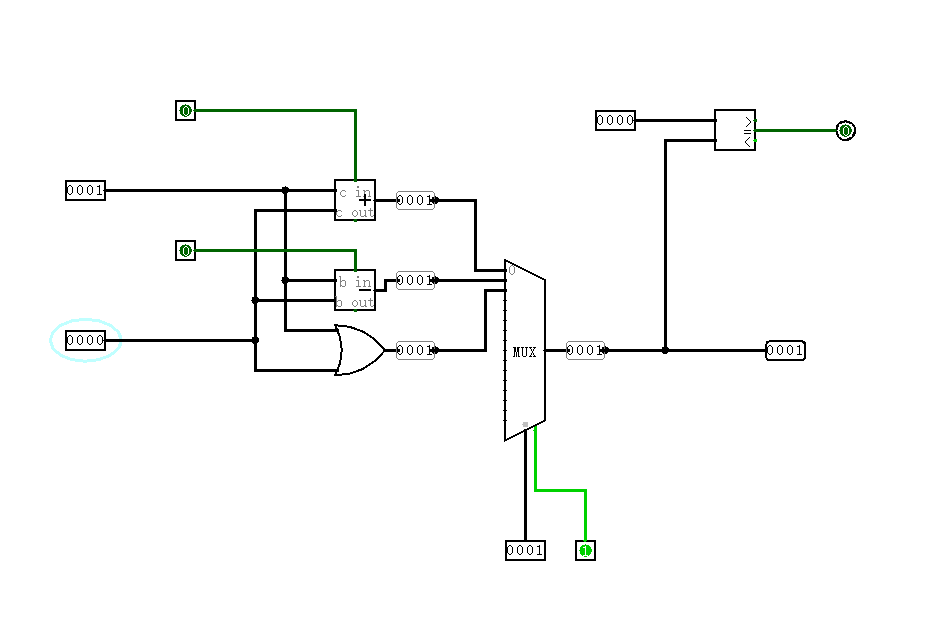


1. 分别设计一个4位、32位的运算器ALU，具有ADDC、SUBB、OR等三种运算功能，具体运算控制参见表3- 1 ALU控制信号定义，并进行仿真测试。

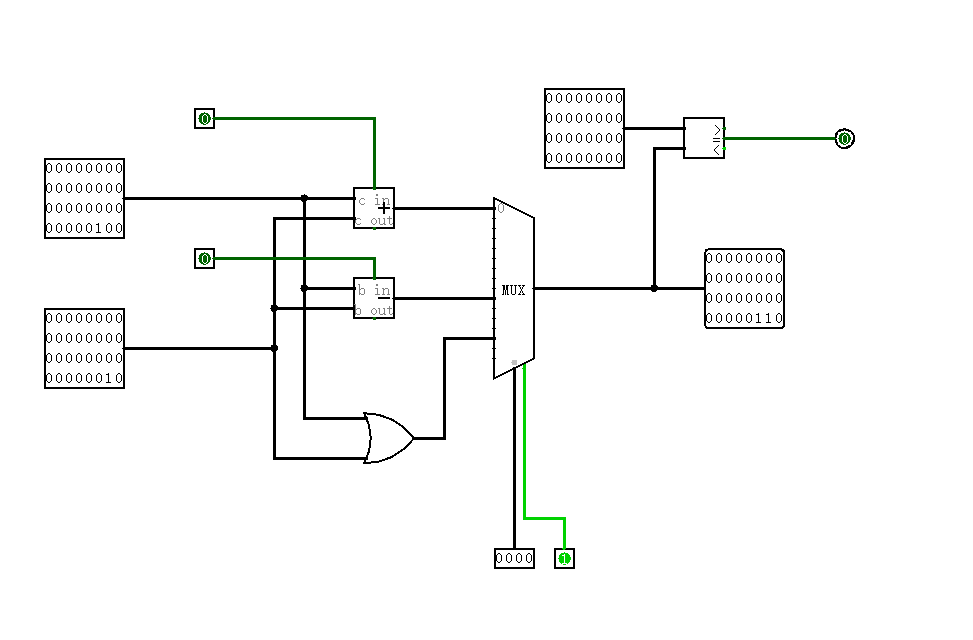
提示：因为ALU的功能是实现两个4位或32位数据的3种运算（加法运算、减法运算、或运算），所以要用到logisim自带的库中元件adder，subtractor，or模块；3种运算的结果哪一路作为ALU的输出，需要一个多路选择器来实现；同时ALU要输出一个运算结果零标志位zero，所以需要一个比较器，将ALU的运算结果与0比较，相等则zero输出为1，否则为0。综合以上分析，共需要添加5种电路（adder，subtractor，or，comparator，multiplier）以及8个输入/输出端点。



（1）4位ALU

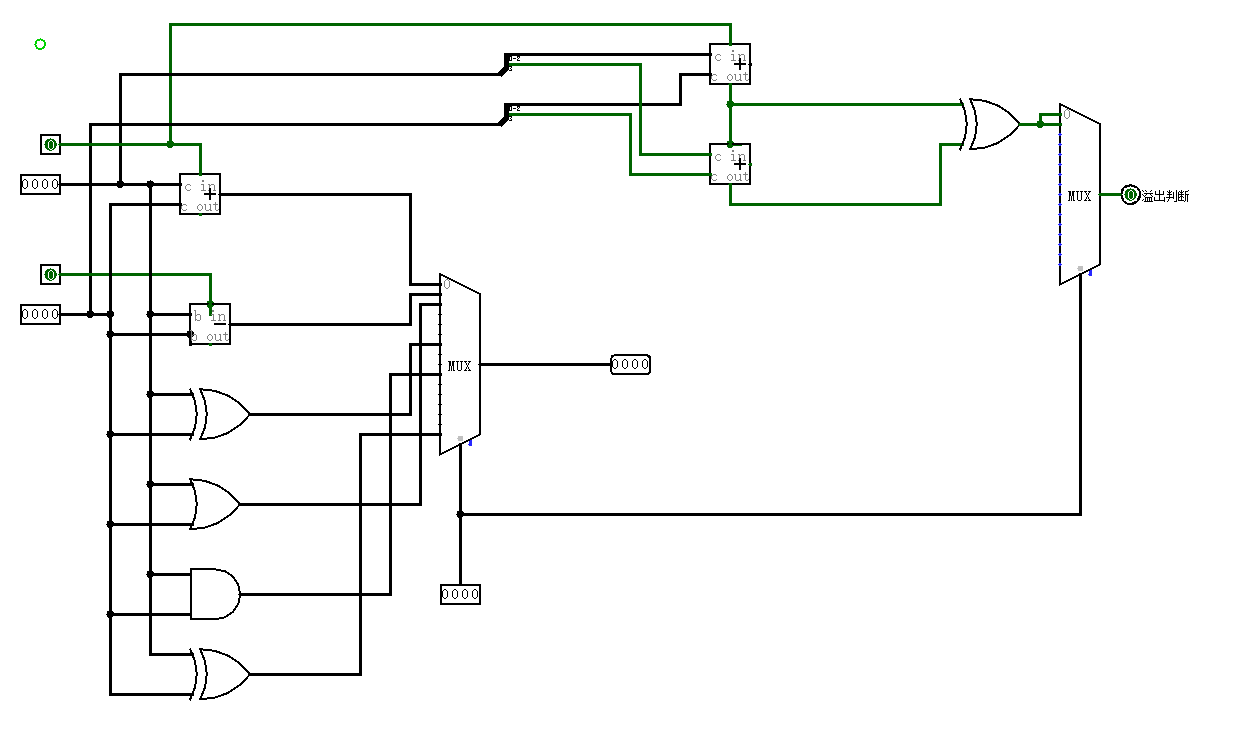


（1）32位ALU



思考题：该运算器支持**与运算**、**异或运算及溢出（over）检测**操作，如何处理？

已4位ALU为例：



与运算、疑惑运算直接添加与逻辑门和异或逻辑门即可。

溢出判断：用分离器将4bit变成3bit（低三位）+1bit（最高位），将3bit+3bit的进位与1bit+1bit的进位异或，若结果为1则溢出。

1. Modelsim实验
2. 用Verilog硬件描述语言实现一个4位的具有基础实验中11种运算功能的运算器ALU，并用Modelsim进行仿真测试。

**原码：**

module ALU(A,B,Cn,OP,F,CLK);

input [3:0] A,B,OP,Cn;

input CLK;

output reg [3:0] F;

always@(posedge CLK or OP)

begin

case(OP)

4'b0000:F = A;

4'b0001:F = A + B;

4'b0010:F = A + B + Cn;

4'b0011:F = A - B;

4'b0100:F = A - B - Cn;

4'b0101:F = A & B;

4'b0110:F = A | B;

4'b0111:F = ~A;

4'b1000:F = B^A;

4'b1001:F = A + 1;

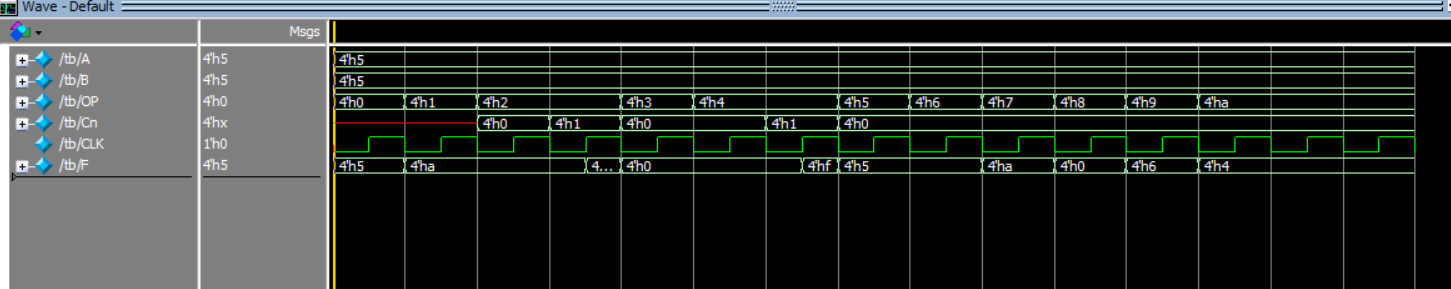
4'b1010:F = A - 1;

endcase

end

endmodule

**测试截图：**



1. 在ALU中增加“同或”、“增4”功能，并进行测试。

**原码：**

module ALU(A,B,Cn,OP,F,CLK);

input [3:0] A,B,OP,Cn;

input CLK;

output reg [3:0] F;

always@(posedge CLK or OP)

begin

case(OP)

4'b0000:F = A; //MOV

4'b0001:F = A + B; //ADD

4'b0010:F = A + B + Cn; //ADDC

4'b0011:F = A - B; //SUB

4'b0100:F = A - B - Cn; //SUBB

4'b0101:F = A & B; //AND

4'b0110:F = A | B; //OR

4'b0111:F = ~A; //NOT

4'b1000:F = A^B; //XOR

4'b1001:F = A + 1; //INC

4'b1010:F = A - 1; //DEC

4'b1011:F = A^~B; //XNOR

4'b1100:F = A + 4; //+4

endcase

end

endmodule

**测试截图：**

