# 实验三 8 位 ALU 设计

ALU 即算术逻辑单元（algorithm logic unit）：简单的来说就是把算术单元和逻辑单元组合在一起。

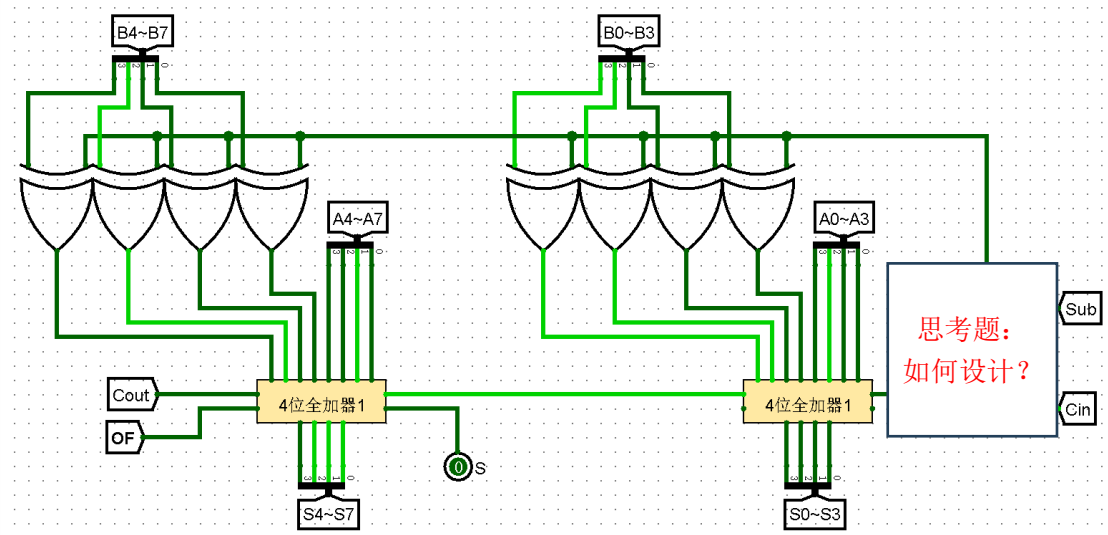
## 1、分析

算术逻辑单元的基本功能包括：算术运算、逻辑运算、移位和比较等，而进行哪种操作可以通过选择器进行选择，所以输入端 A、B 和功能选择端是应该共用的，对于输出端，也应该对移位操作、算术和逻辑操作结果进行选择。

## 2、算术运算

构建算术运算单元，用于实现两个 8 位数据的加减运算，Sub 为减运算标志，OF 为溢出标志，S 为运算结果的最高位，即符号位。

Sub=0，S=A+B；Sub=1，S=A-B




对应的功能表如下所示：

| Sub | Cin | 功能 |
|---|---|---|
| 0 | 0 | A+B |
| 0 | 1 | A+B+1 |

| Sub | Cin | 功能 |
| --- | --- | --- |
| 1 | 0 | A-B |
| 1 | 1 | A-B-1 |

## 3、状态标志

单有数据输入/输出和功能选择还是不够的，很多时候都希望了解运算的状态，如溢出/进位/结果是否为零等，因此还需要添加一些标志的输出。

**状态标志这里设置了溢出标志 OV、符号标志 SI、进位标志 CA、零标志 ZO。**

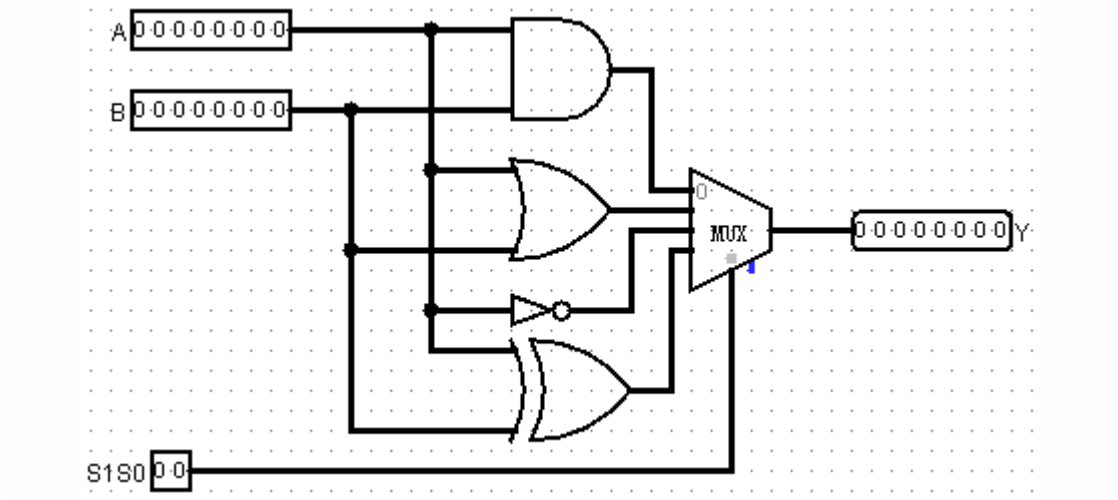
其中 OV 和 SI 只对算术运算有效，CA 对算术和移位运算有效（移出位放入 CA），ZO 对算术和逻辑运算有效。（无效并不意味着输出一定为 0）

溢出的判断方式为最高数值位产生的进位值，与符号位所参生的进位值之间进行异或。

- 溢出 OV 检测：采用了对加法器的符号进位和最高数值进位进行异或运算以实现溢出判定；
- 符号 SG 检测：通过检测结果的最高位即可得出符号，为 1 时表示负数，为 0 表示正数。
- 进位 CA 检测：移位操作移出的位或者是加法器最高位进位，因此需要在不同的操作时选用不同的线路，1 表示进位。
- 零 ZO 检测：当结果为零时，ZO=1；结果不为零，ZO=0。可用或非门实现，所有位都为 0 则或非结果为 1。

## 4、逻辑运算

基本逻辑运算包括与、或、非、异或运算，可通过 4 选 1 选择器选择具体操作。具体实现如下图所示：



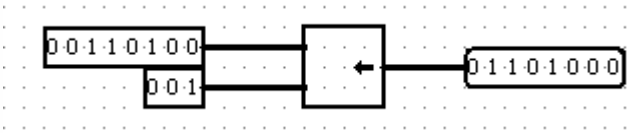

对应的功能表如下所示：

| S1 | S0 | 功能 |
|---|---|---|
| 0 | 0 | A and B |
| 0 | 1 | A or B |
| 1 | 0 | ~A |
| 1 | 1 | A xor B |

## 4、移位器

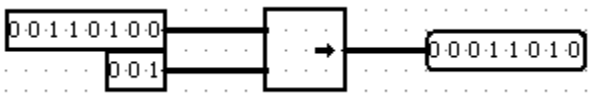
可以直接采用移位寄存器实现。

其中逻辑左移和算术左移操作相同，左移规则是末位补 0，高位移出至 CA 进位标志。

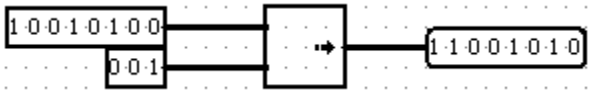


逻辑左移和算术左移

逻辑右移规则是用 0 补充高位，算术右移规则是右移符号位。

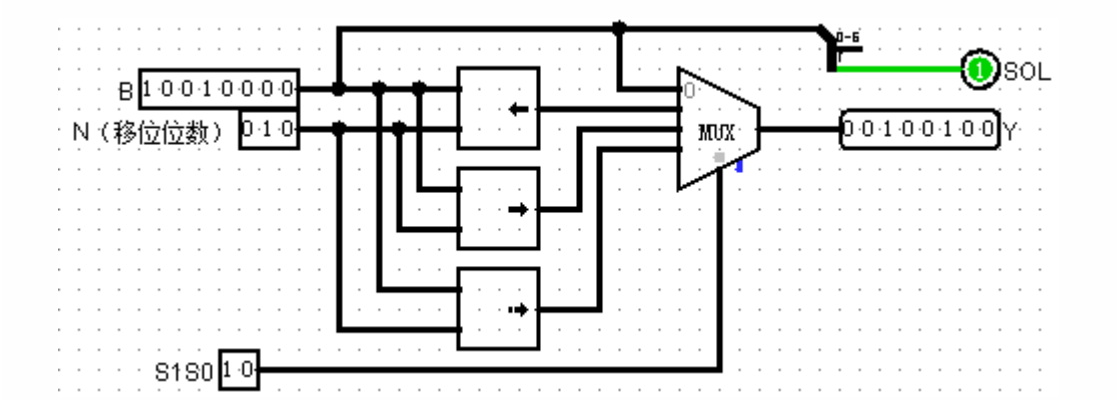


逻辑右移



算术右移

可通过 4 选 1 选择器选择具体的移位操作。如下所示：

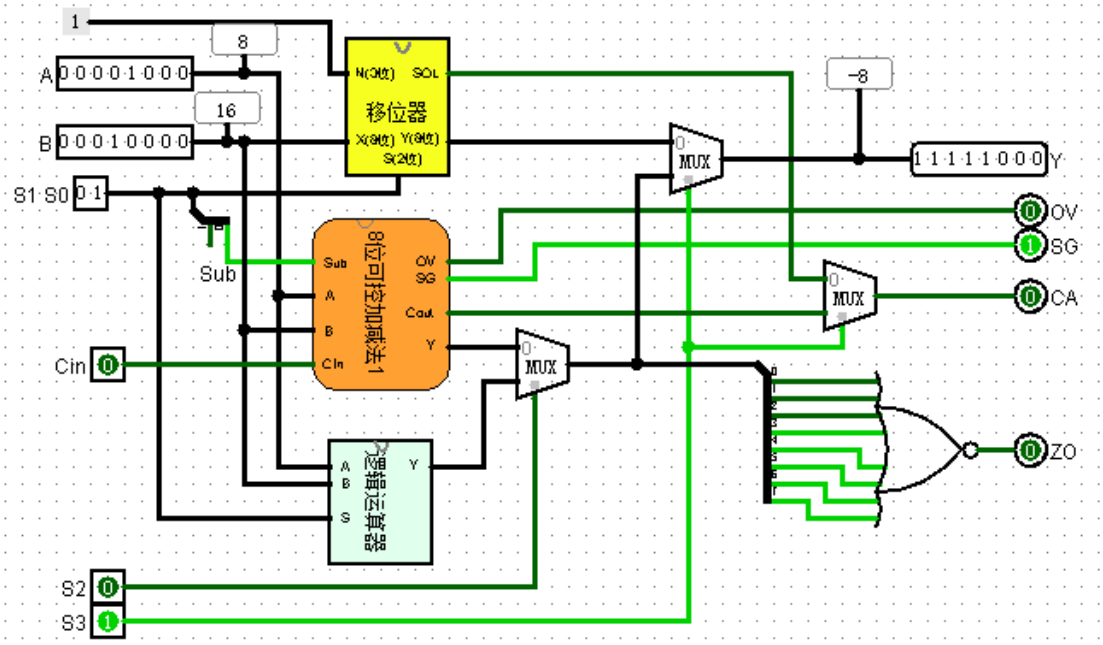



图中 SOL 为左移 1 位时移出至 CA 中的位。对应的功能表如下所示：

| S1 | S0 | 功能 |
|---|---|---|
| 0 | 0 | 传送 B |
| 0 | 1 | 逻辑或算术左移 N 位 |
| 1 | 0 | 逻辑右移 N 位 |
| 1 | 1 | 算术右移 N 位 |

## 6、ALU 设计实现

完成加法器、逻辑运算器、移位器等各芯片的封装。进一步完成 ALU 单元的设计与实现，如下所示：



其功能表如下，请同学认真分析下表，并读懂：

| ALU 操作 | S3 | S2 | S1 | S0(Sub) | Cin | 解释 | 有效标志 |
|---|---|---|---|---|---|---|---|
| 算术运算 | 1 | 0 | 0 | 0 | 0 | A+B（ADD） | OV、SG、CA、ZO |
| | 1 | 0 | 0 | 0 | 1 | A+B+1（ADC） | OV、SG、CA、ZO |
| | 1 | 0 | 0 | 1 | 0 | A-B（SUB） | OV、SG、CA、ZO |
| | 1 | 0 | 0 | 1 | 1 | A-B-1（SBB） | OV、SG、CA、ZO |
| | 1 | 0 | 1 | 0 | 0 | A+1 | |
| | 1 | 0 | 1 | 1 | 0 | A-1 | |
| 逻辑运算 | 1 | 1 | 0 | 0 | x | A and B（AND） | ZO |
| | 1 | 1 | 0 | 1 | x | A or B（OR） | ZO |
| | 1 | 1 | 1 | 0 | x | ~A（NOT） | ZO |
| | 1 | 1 | 1 | 1 | x | A xor B（XOR） | ZO |
| 移位操作 | 0 | x | 0 | 0 | x | MOV B | CA |
| | 0 | x | 0 | 1 | x | 逻辑/算术左移（SHL/SAL） | CA |
| | 0 | x | 1 | 0 | x | 逻辑右移（SHR） | CA |
| | 0 | x | 1 | 1 | x | 算术右移（SAR） | CA |

至此，一个简单的 8 位运算器已经完成。

运算器可以看做一个多功能的加工厂，输入两个数，输出一个结果。要采取哪种功能，必须要通过一些功能选择信号来进行选择。

所以设计时，首先要考虑需要进行哪些运算，然后将这些运算的输入端合并到一起，然后通过选择器选择哪个作为输出，当然也可以采用其他方法，比如用三态门代替数据选择器、通过选择输入从而实现功能选择等，总的原则就是输入-选择功能-输出。

这里没有加入乘法器和除法器，因为这些运算器的体积很大，也会大大增加运算器的复杂性，在现实当中，早期微处理器由于集成度低，规模小，几乎都不设置乘法器。但如果要做乘法怎么办呢？有两种方法，一种是用 CPU 的微指令实现乘法，一种是靠软件编程实现。例如 8086 的 MUL、DIV 实际是通过微指令实现的，CPU 并没有专门的乘法器。但随着集成度的提高，现在的 CPU 都会设

置专门的乘法器或除法器，甚至有的处理器还配置了阵列乘法器。

*选做题：在 ALU 中增加比较器，进一步完善 ALU 的设计，并补充功能表。