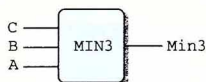


```
!s1      : B; # 01
!s0      : C; # 10
1        : D; # 11
```

];


右边的注释(任何以#开头到行尾结束的文字都是注释)表明了 s1 和 s0 的什么组合会导致该情况会被选中。可以看到选择表达式有时可以简化,因为只有第一个匹配的情况才会被选中。例如,第二个表达式可以写成!s1,而不用写得更完整!s1 && s0,因为另一种可能 s1 等于 0 已经出现在了第一个选择表达式中了。类似地,第三个表达式可以写作!s0,而第四个可以简单地写成 1。


来看最后一个例子,假设我们想设计一个逻辑电路来找一组字 A、B 和 C 中的最小值,如下图所示:



用 HCL 来表达就是:

```
word Min3 = [
    A <= B && A <= C : A;
    B <= A && B <= C : B;
    1                  : C;
];
```

 **练习题 4.11** 计算三个字中最小值的 HCL 代码包含了 4 个形如 $X <= Y$ 的比较表达式。重写代码计算同样的结果,但只使用三个比较。

 **练习题 4.12** 写一个电路的 HCL 代码,对于输入字 A、B 和 C,选择中间值。也就是,输出等于三个输入中居于最小值和最大值之间的那个字。

组合逻辑电路可以设计成在字级数据上执行许多不同类型的操作。具体的设计已经超出了我们讨论的范围。算术/逻辑单元(ALU)是一种很重要的组合电路,图 4-15 是它的一个抽象的图示。这个电路有三个输入:标号为 A 和 B 的两个数据输入,以及一个控制输入。根据控制输入的设置,电路会对数据输入执行不同的算术或逻辑操作。可以看到,这个 ALU 中画的四个操作对应于 Y86-64 指令集支持的四种不同的整数操作,而控制值和这些操作的功能码相对应(图 4-3)。我们还注意到减法的操作数顺序,是输入 B 减去输入 A。之所以这样做,是为了使这个顺序与 subq 指令的参数顺序一致。

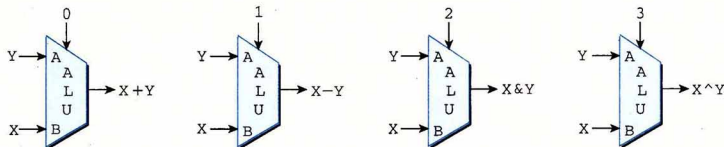


图 4-15 算术/逻辑单元(ALU)。根据函数输入的设置,该电路会执行四种算术和逻辑运算中的一种

4.2.4 集合关系

在处理器设计中,很多时候都需要将一个信号与许多可能匹配的信号做比较,以此来检测正在处理的某个指令代码是否属于某一类指令代码。下面来看一个简单的例子,假设从一个两位信号 code 中选择高位和低位来为图 4-14 中的四路复用器产生信号 s1 和 s0,