数字电路学习笔记（八）：计算电路

JoshCena

好吧，标题不对仗了..... 本文是笔记（七）：经典组合逻辑器件（上）的后续，主要讲两类与计算相关的逻辑电路：加法器与比较器。

一、加法器

1. 半加器

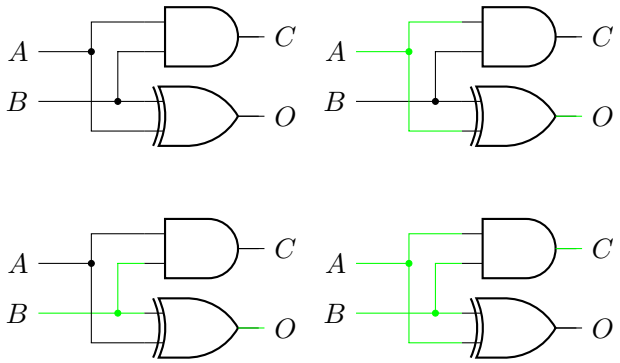
最最基础的加法器，能做一位比特的加法，也就是 $A+B=\overline{CO}$ 这样的形式，（此处的加号是货真价实的“加法”，而不是或逻辑），其中 C 是进位 (carry)。二进制下，加法符合：

$$0+0=00,\quad 0+1=01$$
$$1+0=01,\quad 1+1=10$$

因此列出真值表：

A	B	C	O
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

其实在笔记（三）：基本逻辑中，已经提到过，异或逻辑是二进制下的加法运算——此处我们也可以发现，$O=A\oplus B$。另外还得出 $C=AB$。



这幅图中展示了四种输入情况时不同的输出。

2. 多位加法器

一位加法器并不能使我们满足。如果我们要做的加法不止一位，比如，计算 1101+0111 时，如果是手算，那就会列竖式：

$$\begin{array}{cccccc} & & 1 & 1 & 0 & 1 \\ + & & 0 & 1 & 1 & 1 \\ & & {\scriptstyle 1} & {\scriptstyle 1} & {\scriptstyle 1} & \\ \hline 1 & 0 & 1 & 0 & 0 & \end{array}$$

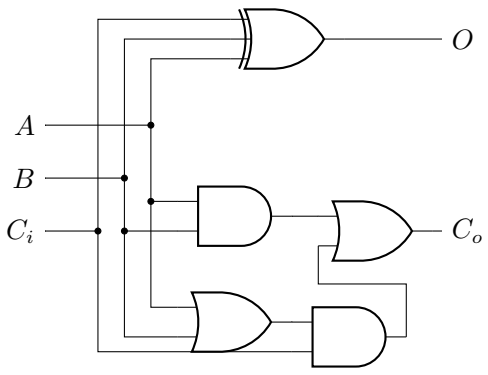
发现其实中间几位在做加法时，需要加三个数字，而不只是两个。考虑一个四比特的加法：

$$\begin{array}{cccccc} & A_3 & A_2 & A_1 & A_0 \\ + & B_3 & B_2 & B_1 & B_0 \\ & (C_2) & (C_1) & (C_0) & \\ \hline C_3 & O_3 & O_2 & O_1 & O_0 \end{array}$$

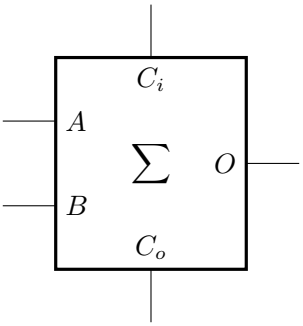
除了最低一位，其他每一位都需要处理三项输入：两个数位和一个来自低位的进位。前一节中的加法器只能有两个输入，只能处理最低一位，对于其他位，我们需要增加一个输入，也就是来自更低位的进位信号。这样，才能实现加法的扩展。完成的运算是：$A + B + C_i = \overline{C_o O}$

A	B	C_i	O	C_o
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$O = A \oplus B \oplus C_i$$
$$C_o = AB + BC_i + AC_i = AB + (A + B)C_i$$



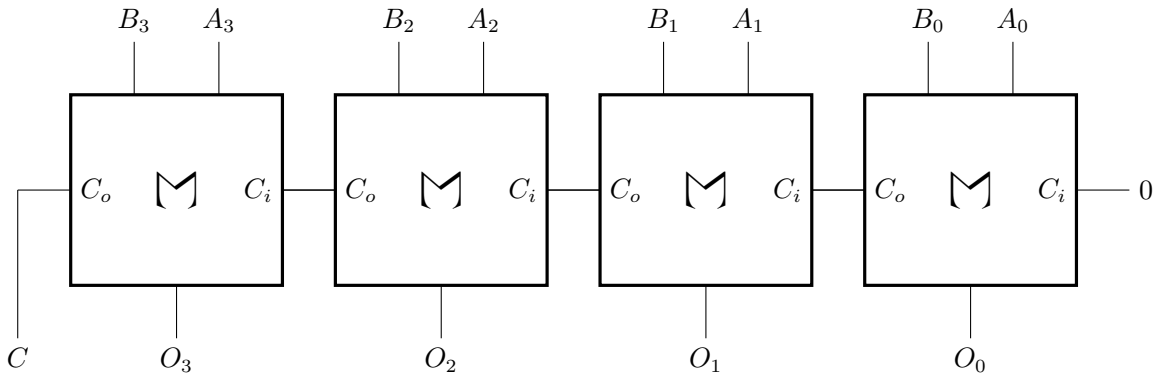
按我的习惯，会像这样表示这个加法器：



这便是一位加法器的抽象。如何将其扩展为更多位的加法器？回到加法的竖式：

$$\begin{array}{cccccc} & A_3 & A_2 & A_1 & A_0 \\ + & B_3 & B_2 & B_1 & B_0 \\ & (C_2) & (C_1) & (C_0) & \\ \hline C_3 & O_3 & O_2 & O_1 & O_0 \end{array}$$

对于每一位输出，将更低位的进位和这一位对应的两个输入相加，然后把进位传递给更高位即可。这是加法器的简单“串联”。注意到，由于在上图的表示中，进位在侧面，连线时就不会和其他的输入数位混淆。



举个例子，输入 $(A_3A_2A_1A_0) = 1101$, $(B_3B_2B_1B_0) = 0111$，则输出 $(CO_3O_2O_1O_0) = 10100$，也就是 13+7=20。

此时设计出的电路就是 **74HC283**（虽然它采用的是超前进位而不是串行进位，但这种技术细节不是非常重要）。它能做四比特的加法——许多计算类的电路都以四比特为单位，因为四比特刚好是一个十进制或十六数字的长度。

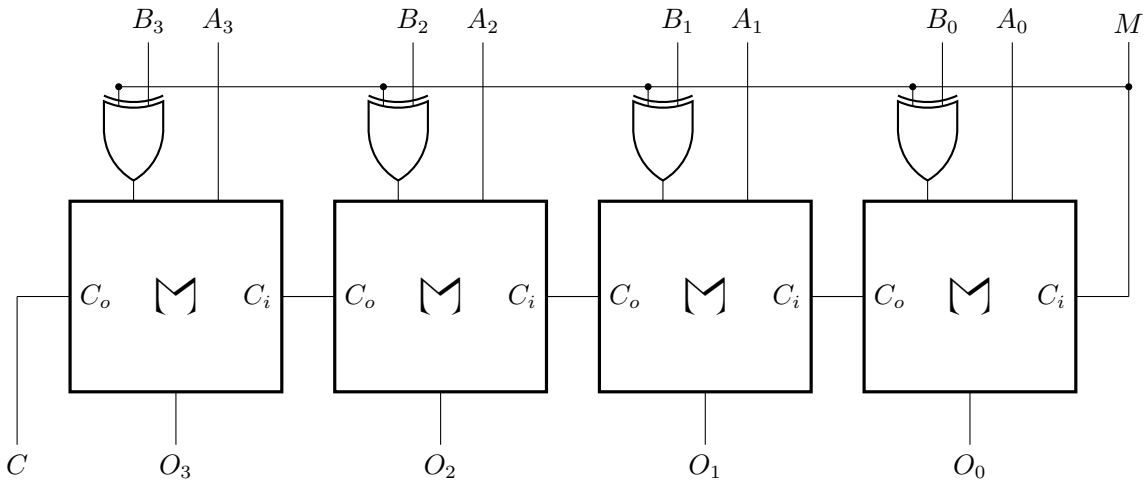
3. 减法器

计算减法，不必再使用新的元件。回顾在笔记（二）：数制与编码中所提到的“补码”：

$$A - B = A + B' + 1$$

所以，如果需要做减法运算，实际上需要做两步：1. 将减数取反；2. 整体 $+1$。其中 $+1$ 一步甚至不需再做一次加法——注意到在加法器中，最低位仍然有一个低位进位 C_i，只是默认被置为了 0。如果将其置为 1，则能很方便地将最终的和加一。因此，在上图中的串行加法器上略作修改，

添加一位输入 M，其为 0 时，电路作加法；值为 1 时，电路作减法。（小技巧：异或门可以用于作原变量-反变量选择器）

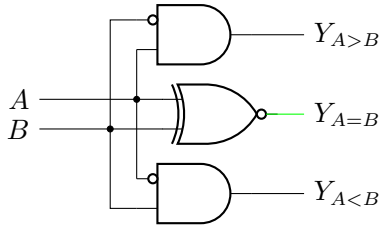


二、比较器

为了做运算，除了做加减法，还要能做比较——比较器能比较两个数的数值大小。两个数 A, B 作比较，有三种情况：$A > B$, $A = B$, $A < B$。每种情况，都用一位输出表示。

A	B	$Y_{A>B}$	$Y_{A=B}$	$Y_{A<B}$
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

$$\begin{cases} Y_{A>B} = AB' \\ Y_{A=B} = A \odot B \\ Y_{A<B} = A'B \end{cases}$$



拓展到更多位数，我们会发现有优先级的存在：如果高位之间已经能比出大小，就没有必要比较更低的位数了——只有在高位相同的情况下，才需要接着比下一位。以比较 $A_3A_2A_1A_0$ 与

$B_3B_2B_1B_0$ 为例，不妨直接写出它的逻辑式：

$$\begin{aligned}Y_{A>B} &= Y_{A_3>B_3} + Y_{A_3=B_3}Y_{A_2>B_2} + Y_{A_3=B_3}Y_{A_2=B_2}Y_{A_1>B_1} \\ &\quad + Y_{A_3=B_3}Y_{A_2=B_2}Y_{A_1=B_1}Y_{A_0>B_0} \\ &= A_3B_3' + (A_3 \odot B_3)A_2B_2' + (A_3 \odot B_3)(A_2 \odot B_2)A_1B_1' \\ &\quad + (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)A_0B_0'\end{aligned}$$

$$\begin{aligned}Y_{A=B} &= Y_{A_3=B_3}Y_{A_2=B_2}Y_{A_1=B_1}Y_{A_0=B_0} \\ &= (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)(A_0 \odot B_0)\end{aligned}$$

$$\begin{aligned}Y_{A<B} &= Y_{A_3<B_3} + Y_{A_3=B_3}Y_{A_2<B_2} + Y_{A_3=B_3}Y_{A_2=B_2}Y_{A_1<B_1} \\ &\quad + Y_{A_3=B_3}Y_{A_2=B_2}Y_{A_1=B_1}Y_{A_0<B_0} \\ &= A_3'B_3 + (A_3 \odot B_3)A_2'B_2 + (A_3 \odot B_3)(A_2 \odot B_2)A_1'B_1 \\ &\quad + (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)A_0'B_0\end{aligned}$$

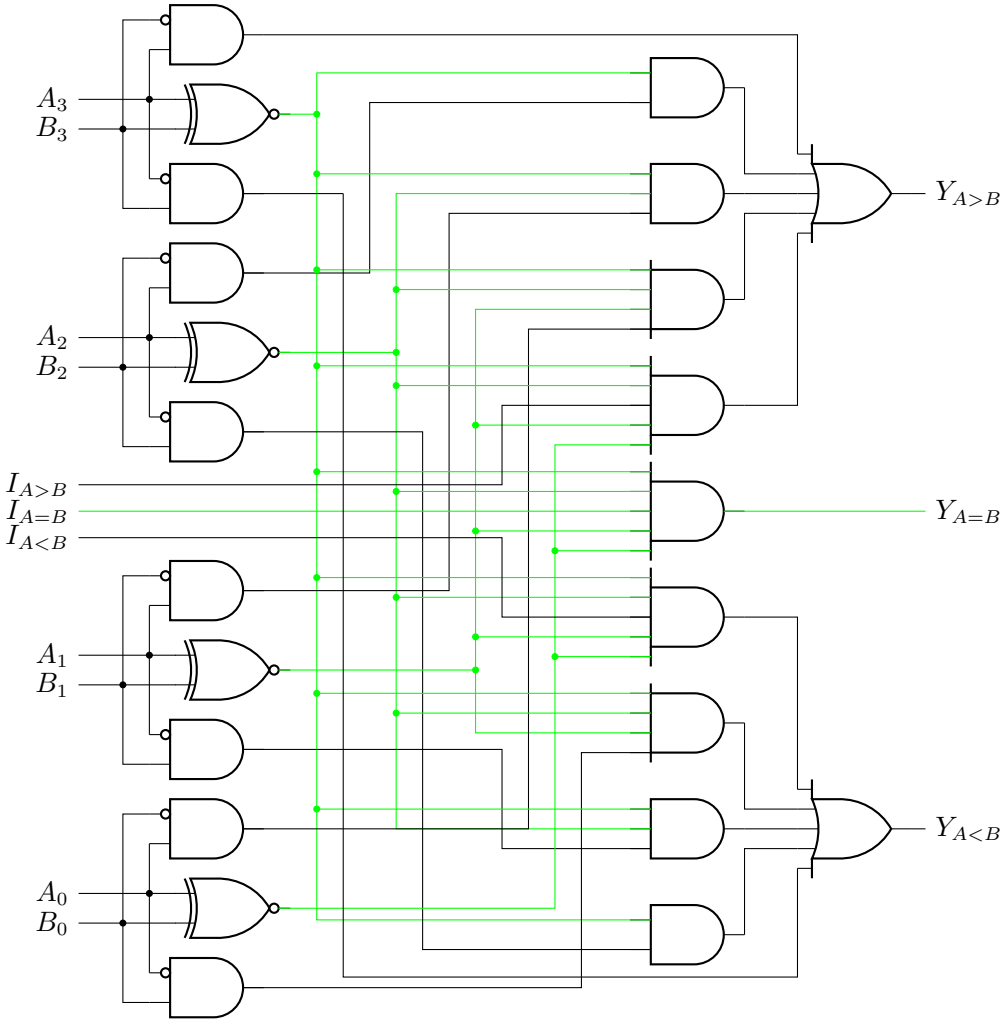
（此公式虽然看着恐怖如斯，但仔细分析，原理非常简单）

同时，和加法器的设计准则一致，为了保证可扩展性，还要再加入一个考虑，就是更低位之间的关系。为此，再加入三个端口：$I_{A>B}$, $I_{A=B}$, $I_{A<B}$。

$$\begin{aligned}Y_{A>B} &= A_3B_3' + (A_3 \odot B_3)A_2B_2' + (A_3 \odot B_3)(A_2 \odot B_2)A_1B_1' \\ &\quad + (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)A_0B_0' \\ &\quad + I_{A>B}(A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)(A_0 \odot B_0)\end{aligned}$$

$$Y_{A=B} = I_{A=B}(A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)(A_0 \odot B_0)$$

$$\begin{aligned}Y_{A<B} &= A_3'B_3 + (A_3 \odot B_3)A_2'B_2 + (A_3 \odot B_3)(A_2 \odot B_2)A_1'B_1 \\ &\quad + (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)A_0'B_0 \\ &\quad + I_{A<B}(A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)(A_0 \odot B_0)\end{aligned}$$



事实上，此电路无论什么时候，$I_{A>B}$，$I_{A=B}$ 和 $I_{A<B}$ 三个输入必有一个为高电平，即使是最低一位，$I_{A=B}$ 也应置为高电平，这样才能正常输出

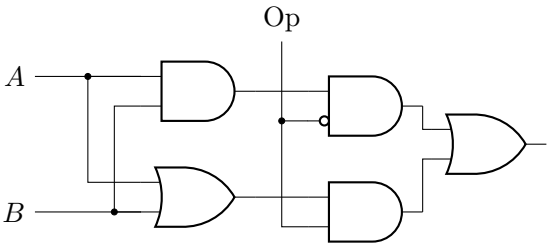
这就是四位比较器 **74HC85** 的内部逻辑。

三、算术逻辑单元（ALU）

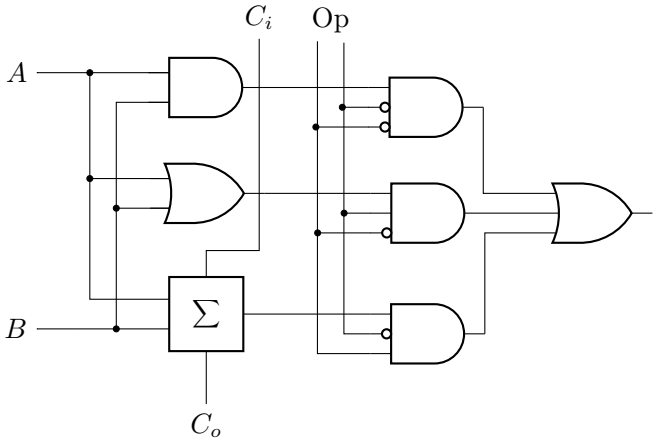
算术逻辑单元（Arithmetic Logic Unit）是计算机中实现数值和逻辑的基础元件——它不是实现某一特定功能的元件，而是许多功能的集合。在前文中实现了所有需要的元件之后，我们可以尝试做出一个 1 比特 ALU。它能执行的指令包括：

```
ADD    X1,  X2,  X3     //加法
SUB    X1,  X2,  X3     //减法
AND    X1,  X2,  X3     //按位与
ORR    X1,  X2,  X3     //按位或
```

每一个指令都要接受两个参数。我们先设计按位与和按位或：每个用一个逻辑门实现，然后再用一个 2 选 1 选择器决定是使用哪个输出。

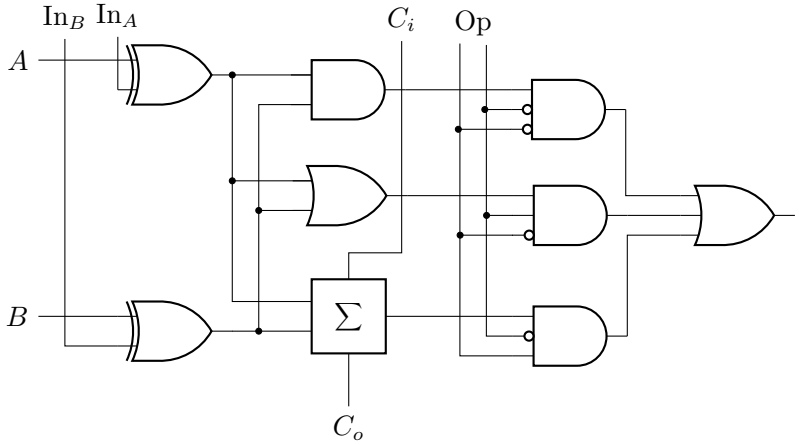


这个电路当操作符 Op 为 0 时，输出 $A \cdot B$；为 1 时，输出 $A + B$。接下来，可以增添加法功能：



把 2 选 1 选择器扩充为了 4 选 1，并且 Op 操作符也变成了两位。Op=00，做`AND A, B`；Op=01，做`ORR A, B`；Op=10，做`ADD A, B`。现在 ALU 也拥有了进位输入输出，可以将其串联，扩展为更多位的运算。

我们继续增添减法功能。由于之前说过，减法不过是将 B 取反，并在最低位的进位输入 1；所以对于每一个 ALU，我们先将 B 取反即可。同时，既然这个电路可以将 B 取反，我们自然也想将 A 取反。这样，我们可以实现更复杂的运算，比如或非运算：$(A + B)' = A'B'$。



其中 In_A 和 In_B 表示了是否对 A 和 B 取反。至此，我们实现了与、或、加、减等几个 ALU 的基本功能。比如，要做减法，应输入 $\mathrm{Op} = 10, \mathrm{In}_A = 0, \mathrm{In}_B = 1$。我们令 ALU 控制指令为 $(\mathrm{In}_A, \mathrm{In}_B, \mathrm{Op})$，则有效指令和对应运算的关系如下：

- 0000: `A & B`

- 0001: `A | B`
- 0010: `A + B`
- 0110: `A - B`

当然，实际的 ALU 的功能不止于此。在未来，会真正设计一个完整的 ALU。

至此，我们介绍完了所有常见的组合逻辑电路，并在这其中反复展示了我们是如何一步步从需求到电路实现的，还演示了几次如何利用现有的集成电路实现更多功能。我们已经有了实现一个计算机所需要的最关键的部分之一——计算单元。从下一章开始，我们将会开始关注 CPU 中另一个灵魂所在——时钟和时序电路。