话题一

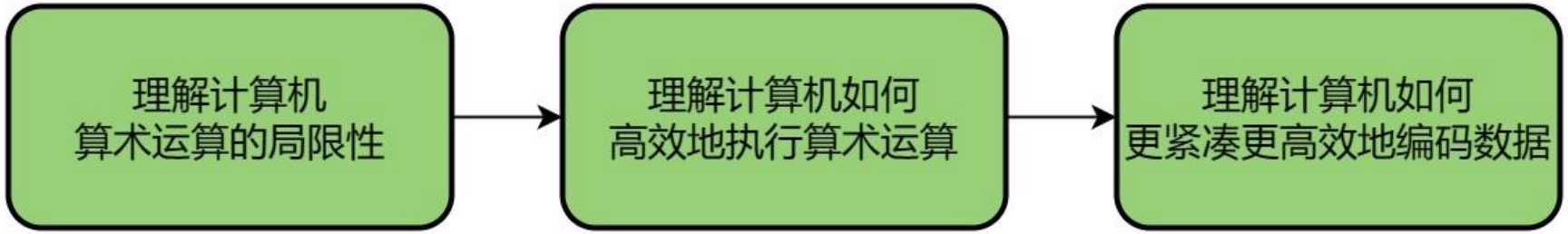
数据的表示，Pt 1

- 薛浩
- stickmind.com

目标

话题 1

计算机是如何表示数值的?



配套练习

Lab 1

- 练习数字的表示、位操作、位掩码操作
- 阅读分析操作位和整型的 C 代码
- 进一步练习 Linux 环境的开发流程

Assignment 1

- 处理加法计算的局限性
- 模拟细胞分裂的过程
- 在终端打印 Unicode 文本

今日话题

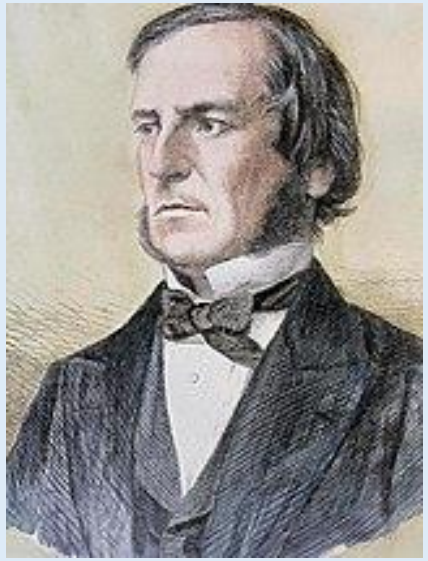
- **Binary**
- Hexadecimal
- Integer Number
  - Unsigned Integer
  - Signed Integer

推荐阅读：

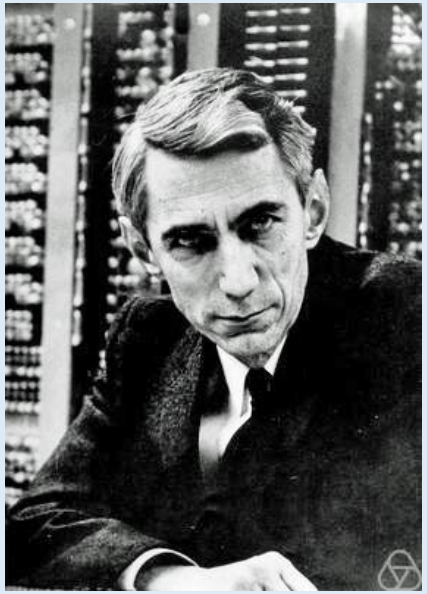
- CSAPP, Ch2

计算机发展的理论基础

- 1854年，英国数学家乔治·布尔（George Boole）发表了一篇具有里程碑意义的论文，详细介绍了后来被称为**布尔代数**的代数逻辑系统。
- 1937年，克劳德·香农（Claude Shannon）在麻省理工学院（MIT）发表了他的硕士论文，该论文在历史上首次使用电子继电器和开关实现了**布尔代数**和**二进制算术**。



George Boole



Claude Shannon

关于 bool

- 早期的 C 语言没有用来表示真和假的值，所以 C 把 0 当作假处理，非 0 当作真处理。
- C99 标准发布后，通过引入头文件 **stdbool.h** 可以使用 bool 类型，允许程序中使用 true 和 false 关键字，但编译器最终还是会转换成 1 和 0 两个值来处理。

```
#include <stdbool.h>

bool flag = true;
flag = false;
```

二进制的优点

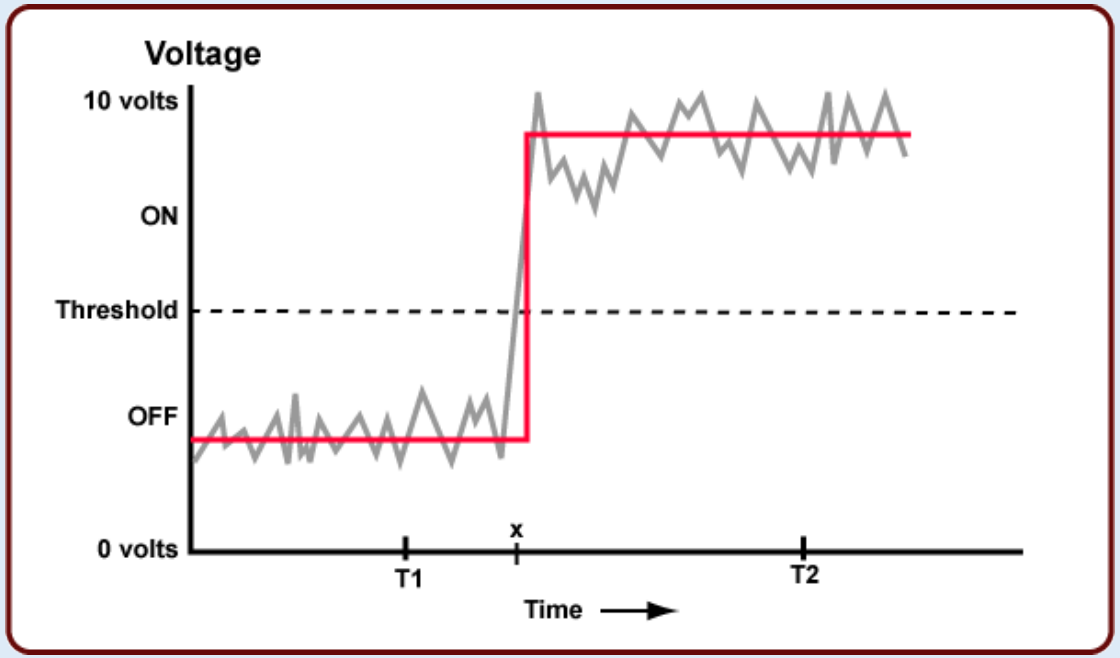
- 二进制设备简单易用

硅芯片容易制造，可以集中在一个小区域

- 二进制信号是明确的，抗噪能力强

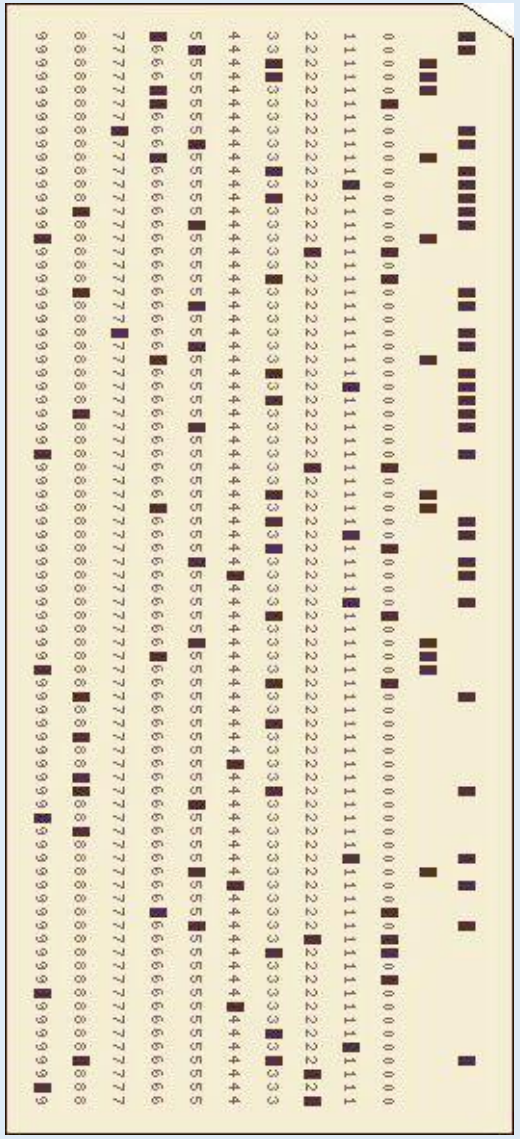
很容易查看数字信号是否开/关。即使有一点噪音，也能够清楚地知道信号是什么

- 可以制作二进制数据的完美副本
- 任何可以用某种模式表示的东西都可以用二进制模式表示



位 Bit

- 位（Bit）是一个单个的 on/off 值，只有这两种可能的结果。
- 位的概念可以理解为 1 或 0、开或关、是或否、真或假的值。
- 位实现的方式有很多种：
  - 电器开关
  - 打孔的纸带
  - 单个晶体管状态
- 位是计算机信息处理的最小单位。



字节 Byte

- 一个位能表示的信息是有限的
- 计算机内存以 8 个位为一组，称为**字节**（Byte）
- 一个字节可以表示 256 个不同的值
- 当某些数据超过 8 位时，可以使用多个字节：
  - int 由 4 个字节（32 位）表示
  - double 由 8 个字节（64 位）表示
  - 颜色通道 RGB 可以用 3 个字节（24 位）表示
- 计算机内存可以看作一个大号**字节数组**（Byte-Addressable）

Name	Number of Bytes	Power of 2
Byte	1	2^0
Kibibyes (KiB)	1024	2^{10}
Mebibytes (MiB)	1,048,576	2^{20}
Gibibytes (GiB)	1,073,741,824	2^{30}
Tebibyte (TiB)	1,099,511,627,776	2^{40}

十进制数字系统

10^3	10^2	10^1	10^0
5	0	2	8

$$5028 = 5 * 10^3 + 0 * 10^2 + 2 * 10^1 + 8 * 10^0$$

二进制数字系统

2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}
1	0	0	1	1	1	0	1	0	0	1	0	0

$$5028 = 1 * 2^{12} + 1 * 2^{9} + 1 * 2^{8} + 1 * 2^{7} + 1 * 2^{5} + 1 * 2^{2}$$

小技巧 基于基数的乘法和除法

- 乘法需要向右移动 1 位并补零：1450 * 10 = 14500 对比 1100 * 2 = 11000
- 除法只需要向左移动 1 位：1450 / 10 = 145 对比 1100 / 2 = 110

今日话题

- Binary
- **Hexadecimal**
- Integer Number
  - Unsigned Integer
  - Signed Integer

推荐阅读：

- CSAPP, Ch2

FAQ 为什么选择十六进制

一般很难准确阅读并记住较大的二进制值。

例如，快速尝试确定 0011100101110110 和 0011100101100110 是否相同。

十六进制可以很方便的表示较大的 0/1 序列，并能够轻松的转换成二进制。

十六进制数字系统

- 十六进制（hexadecimal）是使用 16 个不同的符号表示的数字系统，“0”-“9”表示值 0 到 9，“A”-“F”（或小写）表示 10 到 15 之间的值。

0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
										10	11	12	13	14	15

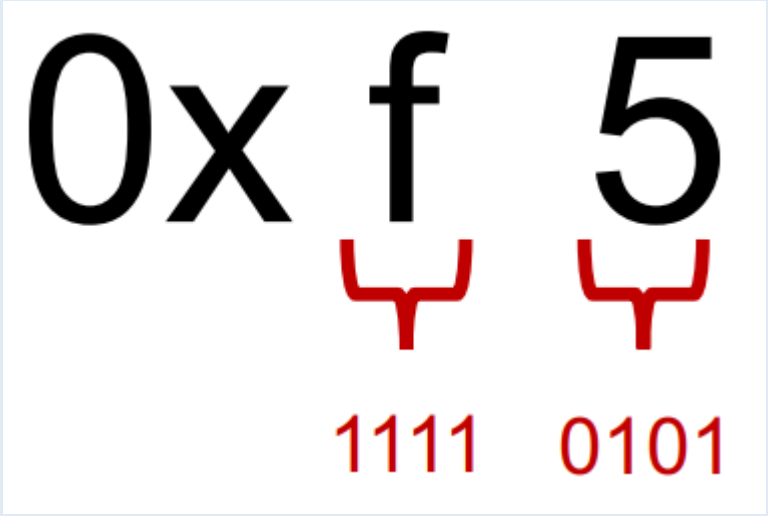
不同进制对应关系

Hex digit	0	1	2	3	4	5	6	7
Decimal value	0	1	2	3	4	5	6	7
Binary value	0000	0001	0010	0011	0100	0101	0110	0111

Hex digit	8	9	A	B	C	D	E	F
Decimal value	8	9	10	11	12	13	14	15
Binary value	1000	1001	1010	1011	1100	1101	1110	1111

区分

- 为了便于区分，十六进制使用 `0x` 开头；二进制使用 `0b` 开头
- 例如，**`0xf5`** 和 **`0b11110101`** 表示同一个值



十进制 vs 二进制 vs 十六进制

165

十进制
- 可读性强
- 不好转换到二进制位模式

0b10100101

二进制
- 计算机使用的格式
- 可读性差

0xa5

十六进制
- 容易转换到二进制位模式
- 可读性好

今日话题

- Binary
- Hexadecimal
- **Integer Number**
  - Unsigned Integer
  - Signed Integer

推荐阅读：

- CSAPP, Ch2

数字的表示

在计算机中，数字的表示可以大致分为三类，分别是：

- 无符号整型（unsigned integer）：表示正数和 0，例如 0，1，2，……，255
- 有符号整型（signed integer）：表示正数、负数和 0，例如 -128，-127，……，0，1，2
- 浮点数（floating point number）：表示实数，例如 3.14，-2.5e-3

今日话题

- Binary
- Hexadecimal
- Integer Number
  - **Unsigned Integer**
  - Signed Integer

推荐阅读：

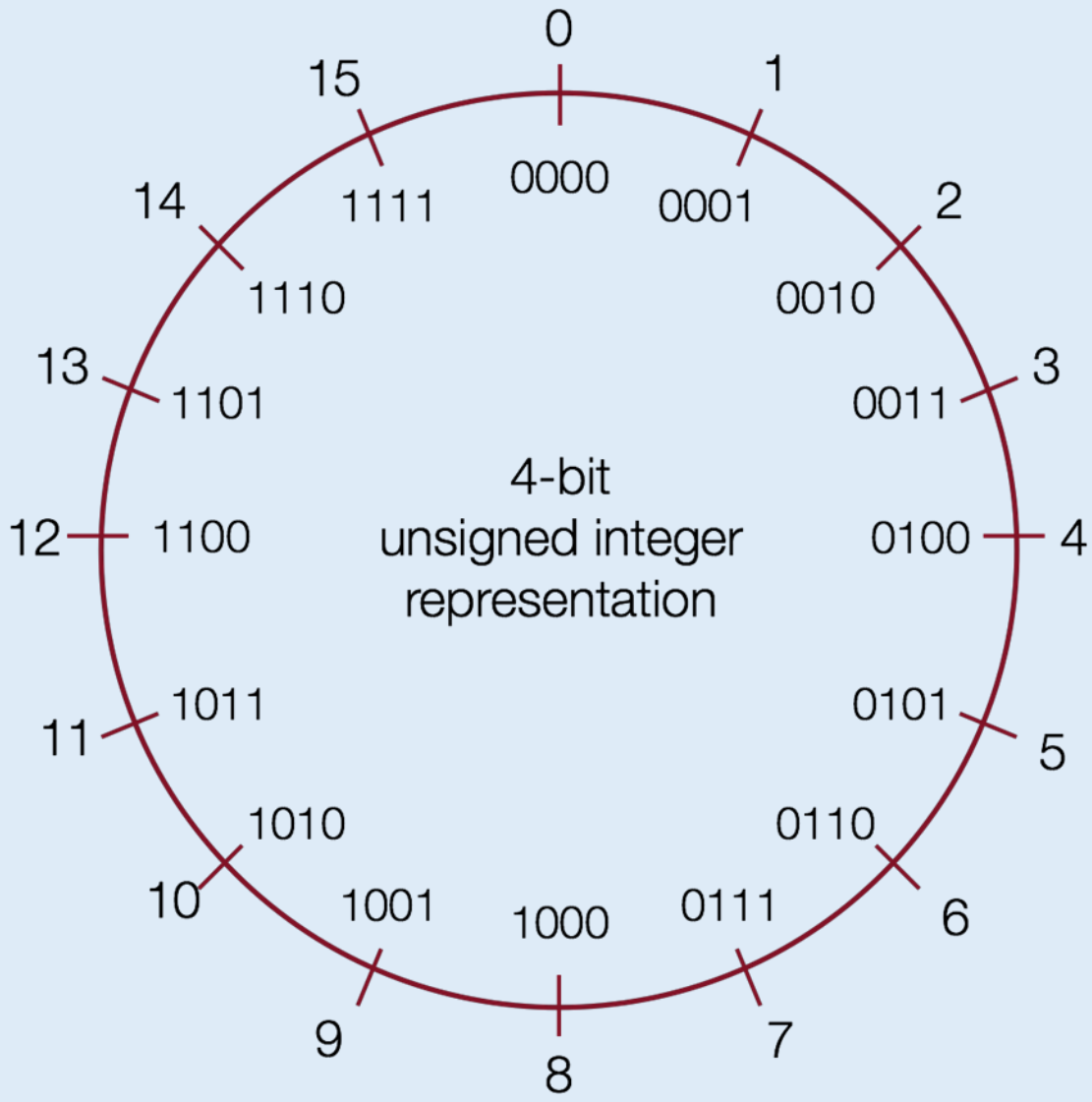
- CSAPP, Ch2

无符号整型

- 无符号整型只能表示 0 和正数，不可以表示负数。
- 无符号整型的二进制表示可以直接映射到十进制数，例如：
  - `0b0001 = 0 * 2^3 + 0 * 2^2 + 0 * 2^1 + 1 * 2^0 = 0 + 0 + 0 + 1 = 1`
  - `0b0101 = 0 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1 * 2^0 = 0 + 4 + 0 + 1 = 5`
  - `0b1011 = 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 1 * 2^0 = 8 + 0 + 2 + 1 = 11`
  - `0b1111 = 1 * 2^3 + 1 * 2^2 + 1 * 2^1 + 1 * 2^0 = 8 + 4 + 2 + 1 = 15`
- 无符号整型表示的范围是 0 → $2^w - 1$，其中 w 表示数据大小
  - 例如 32 位无符号整型可以表示 0 到 $2^{32} - 1$ (4,294,967,295)

无符号整型

- 数字轮盘可以很直观表示无符号整型二进制和十进制之间的关系
- 右图为 4 位无符号整型的表示范围



今日话题

- Binary
- Hexadecimal
- Integer Number
  - Unsigned Integer
  - **Signed Integer**

推荐阅读：

- CSAPP, Ch2

有符号整型

- 有符号整型可以表示 0，正数和负数。
- 既然要表示负数，那么首先需要解决的问题是：如何通过二进制位表示正或负？

有符号整型

- 有符号整型可以表示 0，正数和负数。
- 既然要表示负数，那么首先需要解决的问题是：如何通过二进制位表示正或负？
- 目前的解决方案是

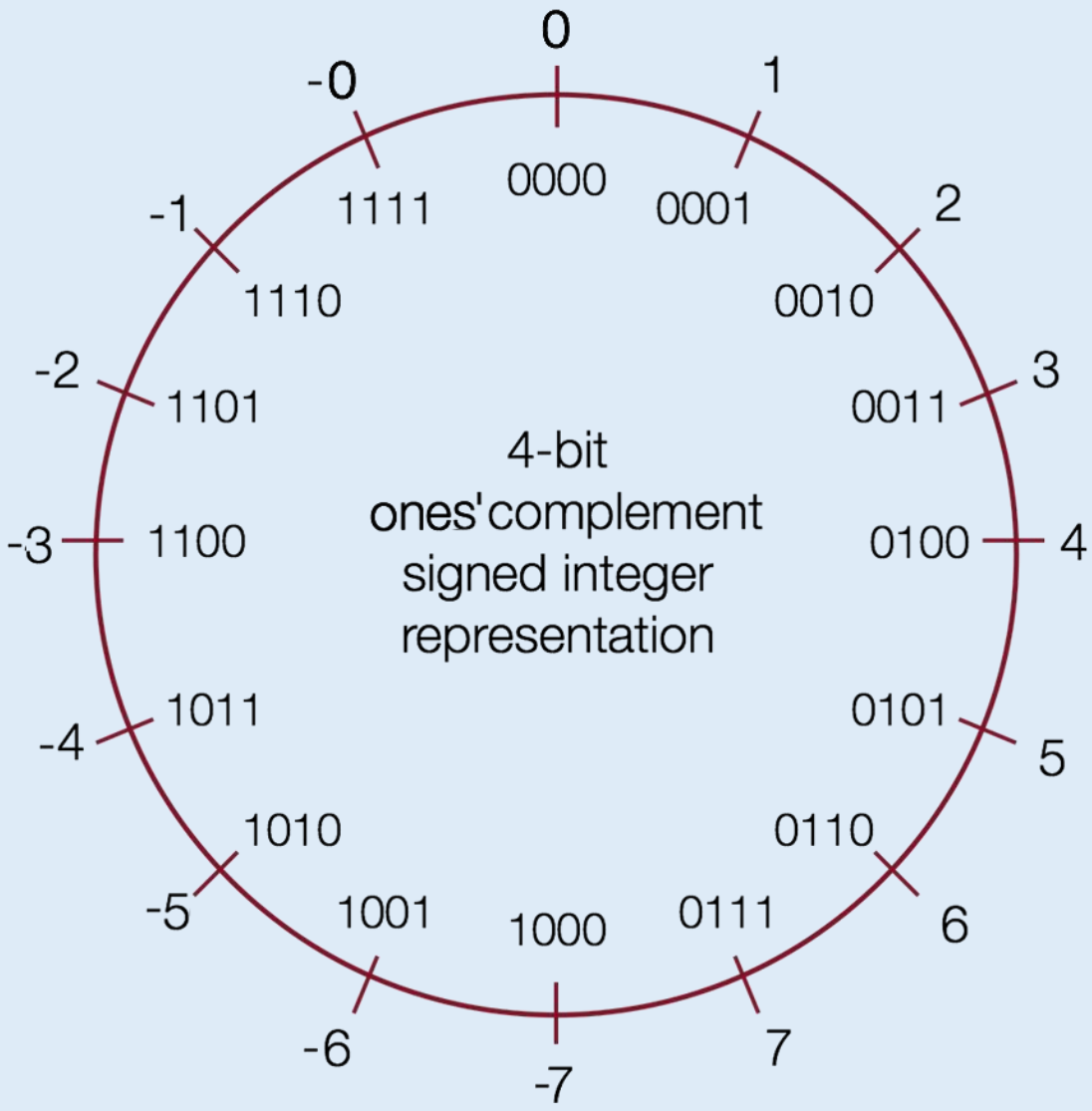
通过保留一个**符号位**（最高位）来区分正和负，

这个位称为**最高有效位** MSB（most significant bit）。

1 的补码 ones' complement

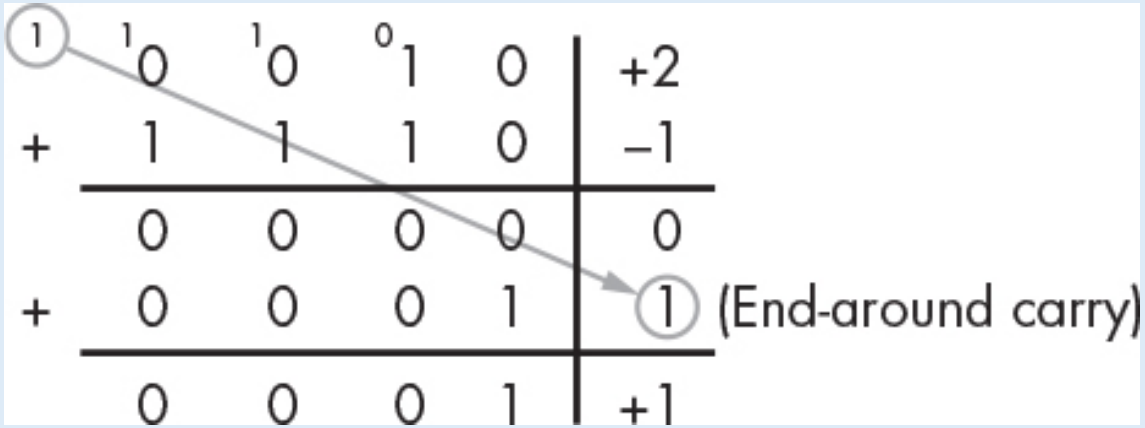
- 通过将正数的所有二进制位进行反转来表示对应的负数，我们可以得到 **1 的补码表示**。
- 这个命名的由来是基于这样的一个事实，对应的正负数相加可以得到**全为 1 的二进制位**。

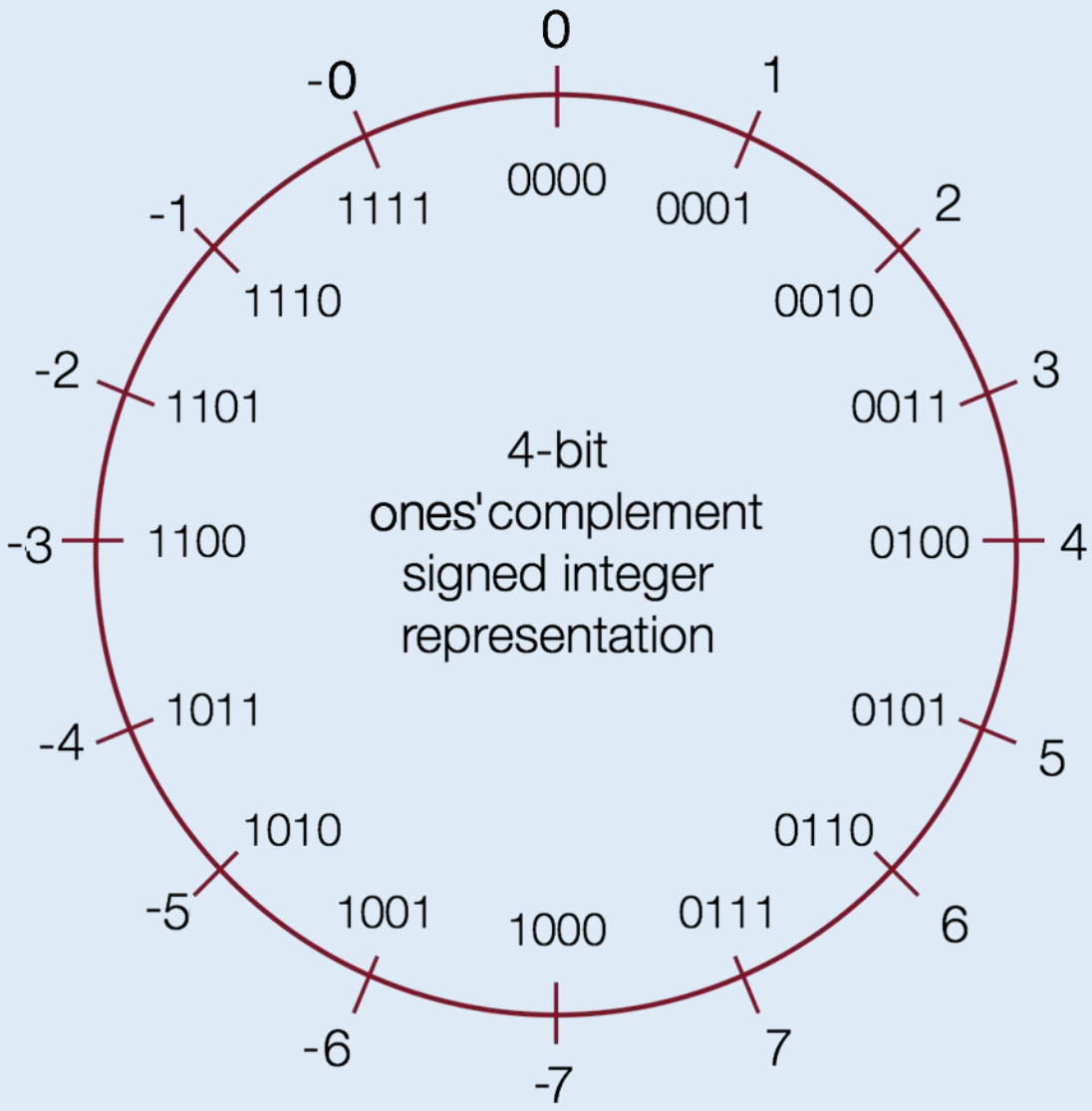
  例如 -1 和 1 相加的二进制位是 1111。
- 目前，1 的补码表示在**数字信号处理**领域依然有大量的应用。



缺点

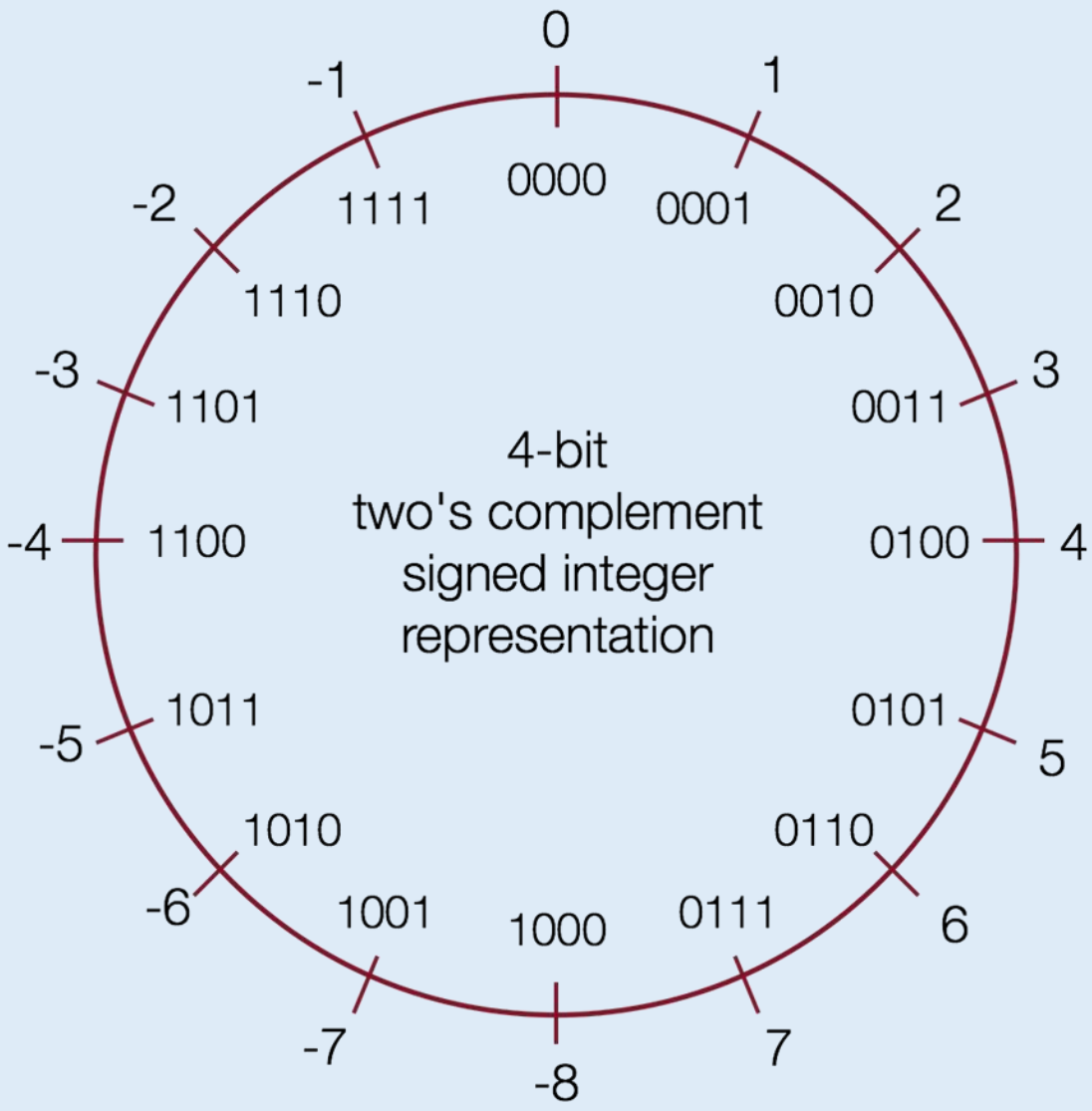
- 有两个用于表示 0 的不同位模式，造成了不必要的浪费。
- 在进行加法操作时，如果进位超过了总体位数，则需要进行**循环进位**（end-around carry）。





2 的补码 two's complement

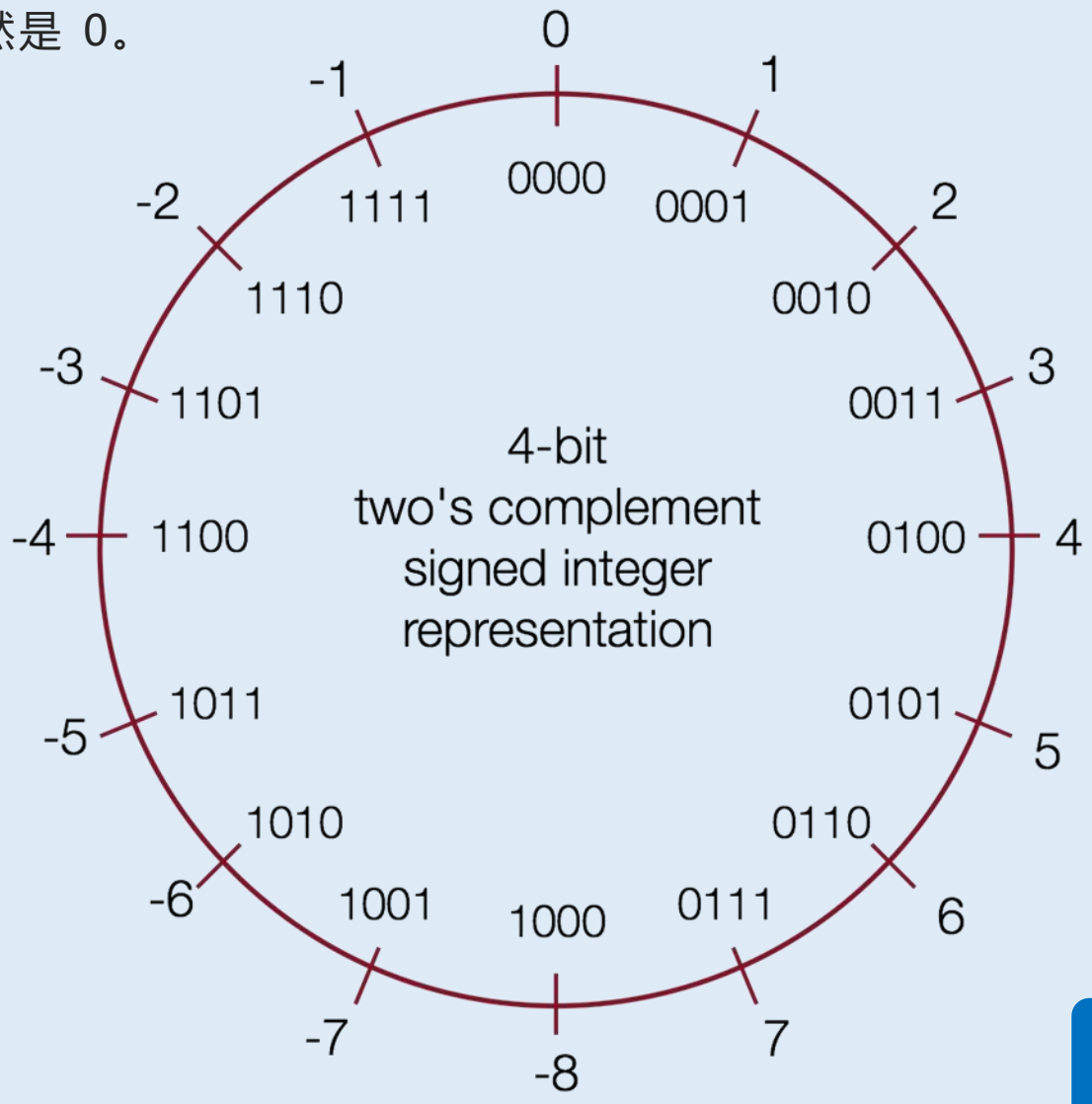
- 如果我们想避免循环进位，可以对反转后的位额外进行一次加 1 操作，那么我们就得到了 **2 的补码表示**。
- 例如，+1 二进制表示为 0001，反转后的结果是 1110，然后再进行加 1 操作得到 1111 用于表示 -1；
- 同样，+2 二进制表示为 0010，反转后的结果是 1101，然后再进行加 1 操作得到 1110 用于表示 -2。



优点

- 只有一个位模式表示 0，对应的正负数相加的结果依然是 0。

```
  0101 (+5)          1111  (-1)
+ 1011 (-5)        + 0001  (+1)
  ----               ----
  0000 (0)           0000  (0)
```



优点

- 只有一个位模式表示 0，对应的正负数相加的结果依然是 0。
- 只有一个位模式用于表示 0，避免了浪费。
- 最高有效位 MSB 依然可以用于表示正负号。

```
  0010 (+2)          0100  (+4)              0100  (+4)
+ 1011 (-5)        - 0101  (+5)    ->    + 1011  (-5)
  ----               ----                  ----
  1101 (-3)          ????  (??)            1111  (-1)
```

优点

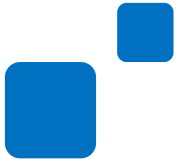
- 只有一个位模式表示 0，对应的正负数相加的结果依然是 0。
- 只有一个位模式用于表示 0，避免了浪费。
- 最高有效位 MSB 依然可以用于表示正负号。
- 加法操作逻辑更简单，可以用于任意的正负数组合，不需要循环进位。

```
  0010 (+2)          0100  (+4)              0100  (+4)
+ 1011 (-5)        - 0101  (+5)    ->    + 1011  (-5)
  ----               ----                  ----
  1101 (-3)          ????  (??)            1111  (-1)
```

优点

- 只有一个位模式表示 0，对应的正负数相加的结果依然是 0。
- 只有一个位模式用于表示 0，避免了浪费。
- 最高有效位 MSB 依然可以用于表示正负号。
- 加法操作逻辑更简单，可以用于任意的正负数组合，不需要循环进位。

```
  0010 (+2)          0100  (+4)              0100  (+4)
+ 1011 (-5)        - 0101  (+5)    ->    + 1011  (-5)
  ----               ----                  ----
  1101 (-3)          ????  (??)            1111  (-1)
```



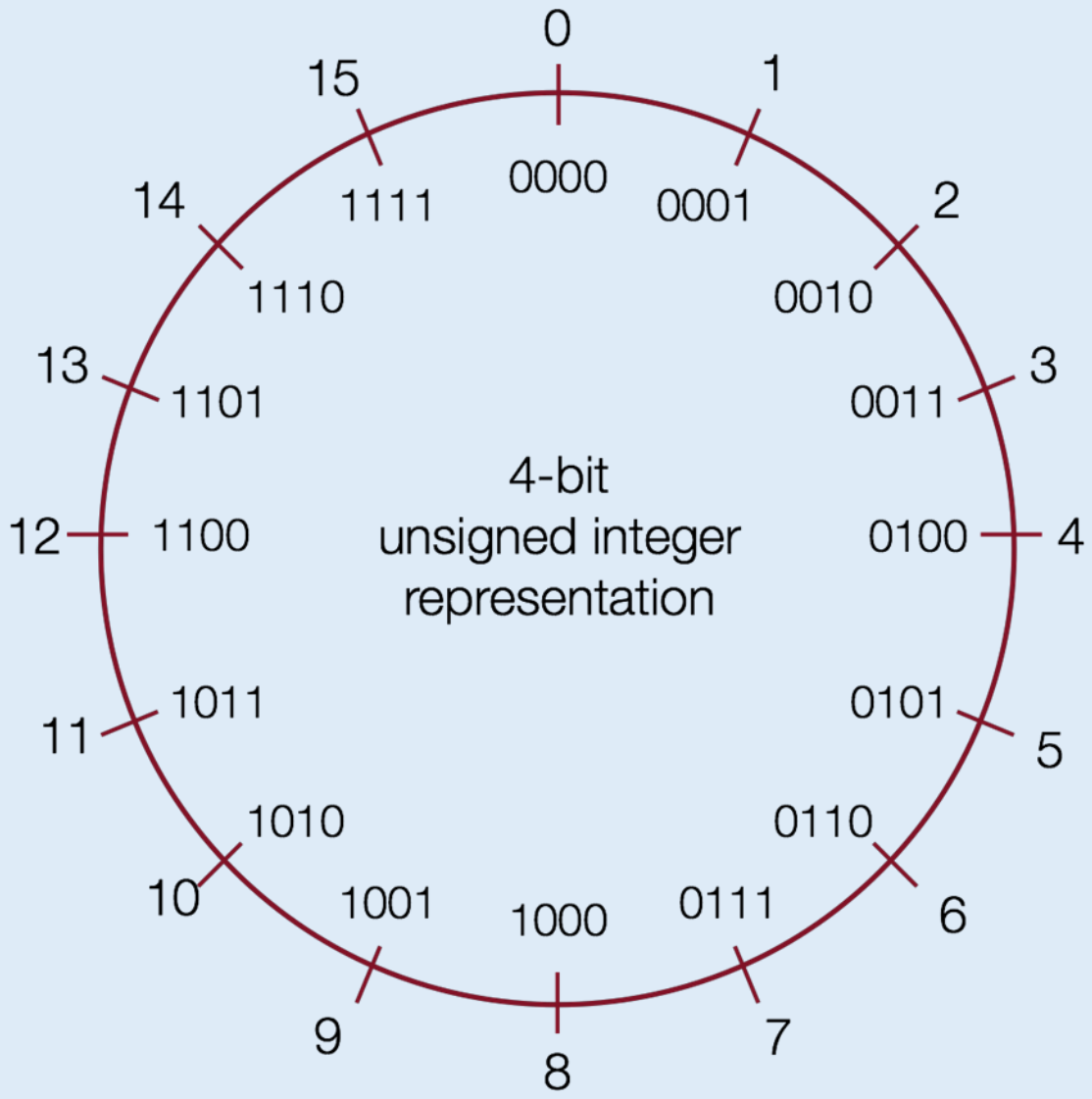
小技巧 如何快速找到其对应的正/负值？

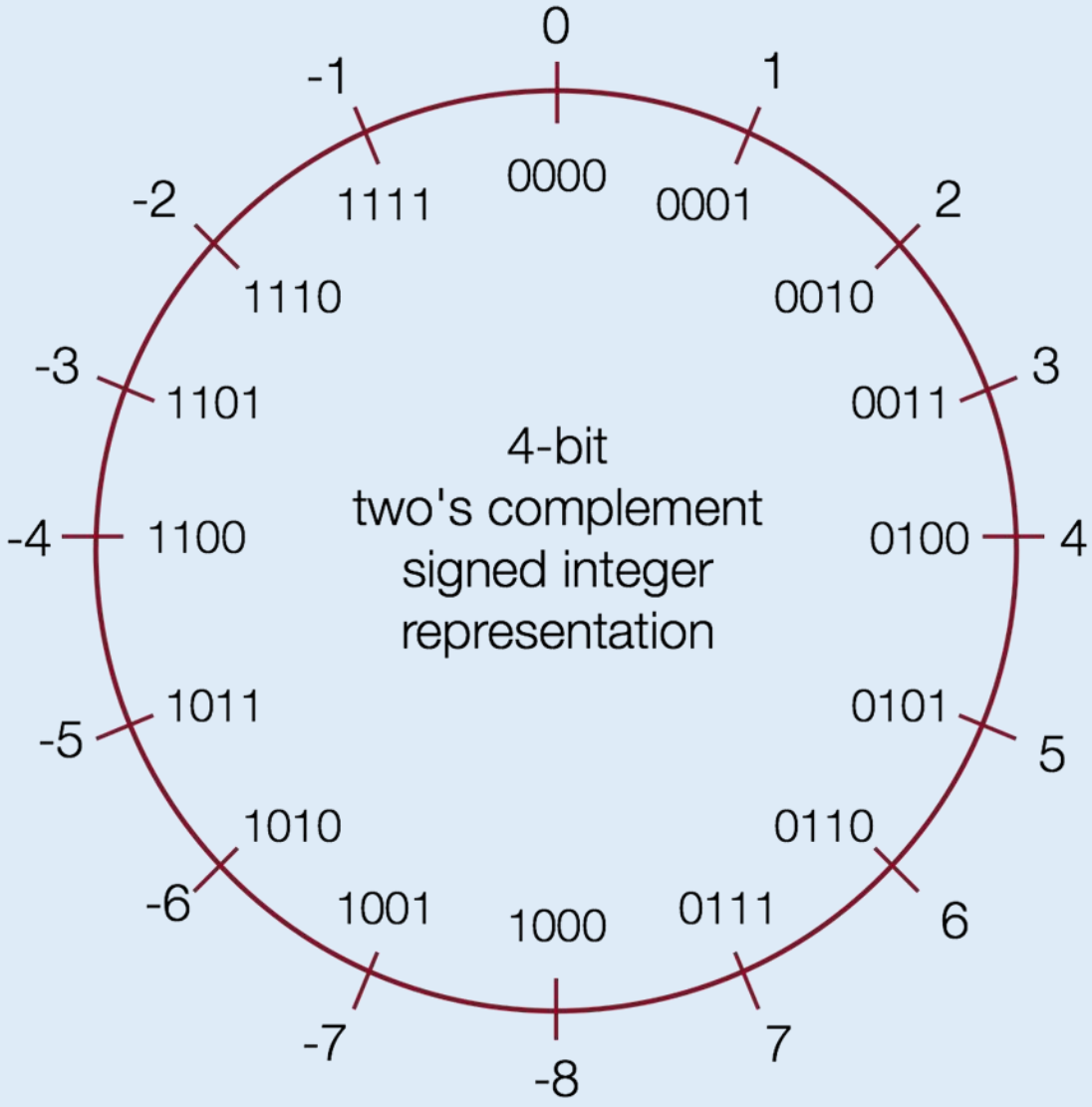
从右向左，找到第一个 1，将左侧剩下的位全部反转即可。

例如，+2 的位模式为 0010，那么将第一个 1 左侧所有位反转可以得到 1110，即为 -2。

推荐阅读：CSAPP 2.3.3 补码的非

无符号 vs 有符号


0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
4-bit
unsigned integer
representation


0
1
2
3
4
5
6
7
-8
-7
-6
-5
-4
-3
-2
-1
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
4-bit
two's complement
signed integer
representation

问题？