话题——

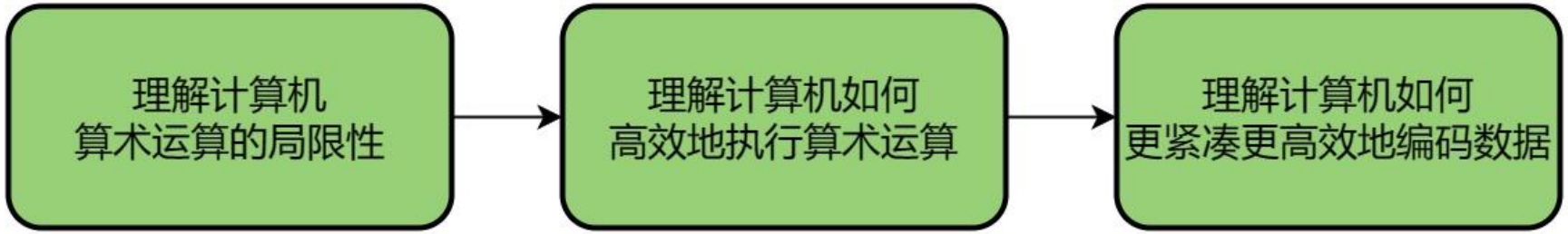
数据的表示，Pt 3

- 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

常见操作符

- 基于字节级别的操作符
  - **Arithmetic operators:** +, -, *, /, %
  - **Comparison operators:** ==, !=, <, >, <=, >=
  - **Logical Operators**: &&, ||, !
- 基于位级别的操作符：
  - **bitwise operators:** &, |, ~, ^, <<, >>

今日话题

- **Bitwise Boolean Operation**
- Bitwise Shift Operation
- Bitmask

推荐阅读：

- CSAPP, Ch2.1
- K&R, Ch2.9

布尔逻辑

- 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。
- 在计算机中，true 编码为 1，false 编码为 0

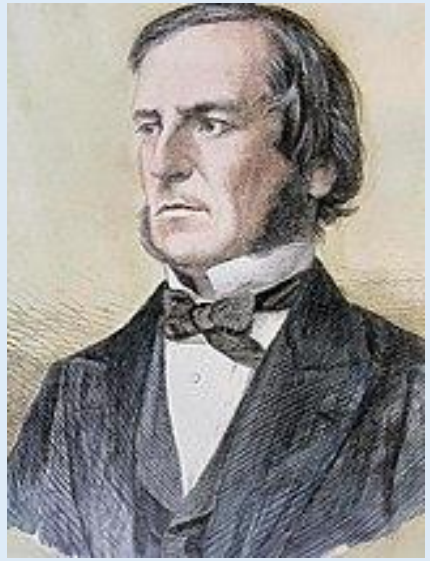
~	
0	1
1	0

&	0	1
0	0	0
1	0	1

|	0	1
0	0	1
1	1	1

^	0	1
0	0	1
1	1	0

Figure 2.7 **Operations of Boolean algebra.** Binary values 1 and 0 encode logic values TRUE and FALSE, while operations ~, &, |, and ^ encode logical operations NOT, AND, OR, and EXCLUSIVE-OR, respectively.



George Boole

位运算

AND	OR	XOR	NOT
0110	0110	0110	
& 1100	| 1100	^ 1100	~ 1100
----	----	----	----

辨析 逻辑运算符

- 在 C 语言中，逻辑运算（||，&&，!）很容易和位运算混淆。逻辑运算认为非 0 为 true，0 为 false。
- 逻辑运算是按**整体数值**来计算结果的，并不是在**位级别**上进行操作；这是区别于位运算最大的地方。

逻辑运算

```
    AND          OR           NOT

    0110         0110

&& 1100      || 1100       ! 1100

    ----         ----         ----
```

逻辑操作

AND	OR	NOT
true	true	
&& true	|| true	! true
----	----	----

问题 7

bits_playground.c

今日话题

- Bitwise Boolean Operation
- **Bitwise Shift Operation**
- Bitmask

推荐阅读：

- CSAPP, Ch2.1
- K&R, Ch2.9

移位操作

- 移位运算也可以称为一种位运算，因为该操作符将值当作一系列的位来对待，而不是一个整体数值。
- 寄存器的位数是固定的，在移位过程中，总会有一些位被移出（shifted out）寄存器，也会有一些位被移进（shifted in）寄存器。

```
x << k;  // 该表达式等于将 x 向左移动 k 个位后的值（x 不变）

x >>= k; // 将 x 向右移动 k 个位（x 改变）
```

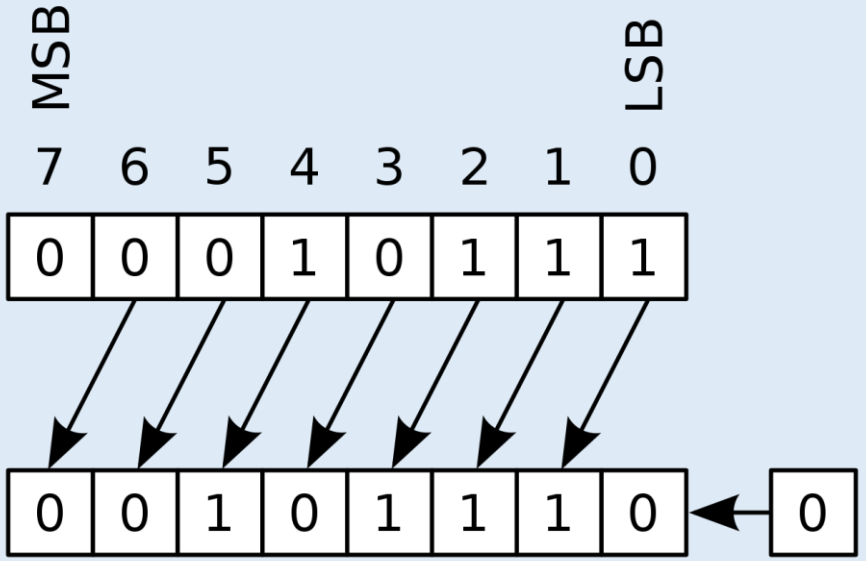
- 在 C 语言中，k 为**负值**或大于等于**数据类型宽度**是**未定义行为**（Undefined Behavior）

```
x << 32; // error

x >> 36; // error

x << 40; // error
```

左移 Left Shift

- 将二进制位模式向左移动指定数量的位置。移动之后，低位（lower order）增加的部分**补充 0**，高位多余的部分将会丢失。

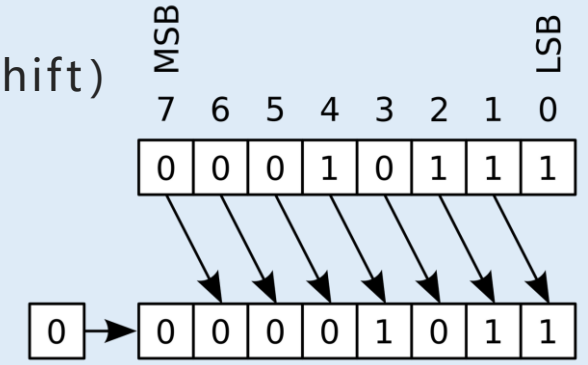
```
00110111 << 2 // 11011100

01100011 << 4 // 00110000

10010101 << 4 // 01010000
```



右移 Right Shift

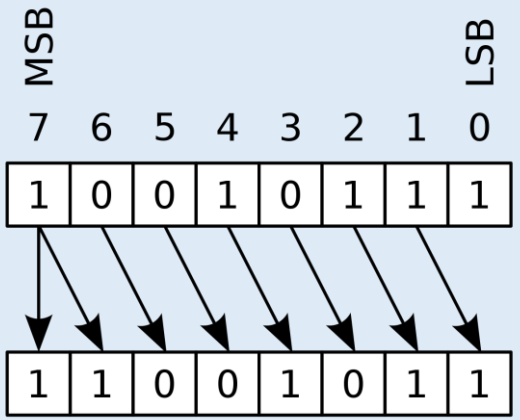
- 将二进制位模式向右移动指定数量的位置。移动之后，低位多余的部分将会丢失，但是高位（higher order）并不会默认补充 0。
- 对于**无符号整型**，高位**补充 0**，这种右移操作称为**逻辑右移**（Logical Right Shift）

```
unsigned short ux = 16;      // 0000 0000 0001 0000
unsigned short uy = x >> 4; // 0000 0000 0000 0001
```



- 对于**有符号整型**，高位**补充符号位**，这种右移操作称为**算术右移**（Arithmetic Right Shift）

```
short x = 16;      // 0000 0000 0001 0000
short y = x >> 4; // 0000 0000 0000 0001
x = -16;     // 1111 1111 1111 0000
y = x >> 4; // 1111 1111 1111 1111
```



陷阱 操作符的优先级

- 由于加法操作符优先级比移位运算符高，所以表达式 `1<<2 + 3<<4` 在 C 语言中的解释是

  `1 << (2+3) << 4`

- 为了避免不必要的麻烦，尽量多加一些括号，比如这样 (1<<2) + (3<<4)。

参考阅读: K&R Ch2.11

陷阱 整型字面量

- 对于语句 `long num = 1 << 32;`，由于字面量默认是 int 类型，最多只有 32 个位，所以 1 << 2 是未定义行为。
- 需要将3上述语句修改为 long num = 1**L** << 32; 才会正常编译执行。

关键思想

- 算术移位补充**符号位**
- 逻辑移位补充**0**

问题 7

shift.c

今日话题

- Bitwise Boolean Operation
- Bitwise Shift Operation
- **Bitmask**

推荐阅读：

- CSAPP, Ch2.1
- K&R, Ch2.9

应用：位掩码

- 位操作在嵌入式领域应用较为广泛
- 对一组位模式进行**选择性操作**，即屏蔽掉部分位，只对感兴趣的部分进行修改

位掩码 Bitmask

- 对一组位模式进行**选择性操作**，即屏蔽掉部分位，只对感兴趣的部分进行修改，这种操作称为**位掩码**（Bitmask）。
- **将指定的位设置为 1，即置 1 操作**
  - Y | 1 = 1 按位或运算 | 搭配 1 可以将位设置为 1
  - Y | 0 = Y 按位或运算 | 搭配 0 可以保持位状态不变

```
  10010101    10100101
| 11110000    00010000
  --------    --------
  11110101    10110101
```

以上示例也可以理解为，求两组位模式的**并集**。

位掩码 Bitmask

- 对一组位模式进行**选择性操作**，即屏蔽掉部分位，只对感兴趣的部分进行修改，这种操作称为**位掩码**（Bitmask）。
- **将指定的位设置为 0，即置 0 操作**
  - Y & 1 = Y 按位与运算 & 搭配 0 可以将位设置为 0
  - Y & 0 = 0 按位与运算 & 搭配 1 可以保持位状态不变

```
  10010101    10100101
& 00001111    11001111
  --------    --------
  00000101    10000101
```

以上示例也可以理解为，求两组位模式的**交集**。

位掩码 Bitmask

- 对一组位模式进行**选择性操作**，即屏蔽掉部分位，只对感兴趣的部分进行修改，这种操作称为**位掩码**（Bitmask）。
- **查询某个位的状态**
  - 按位与运算 & 稍作变化就可以进行查询操作
  - 结果为 true，则查询的位是 1；结果为 false，则查询的位为 0

```
  10011101    10010101
& 00001000    00001000
  --------    --------
  00001000    00000000    -> 布尔值
```

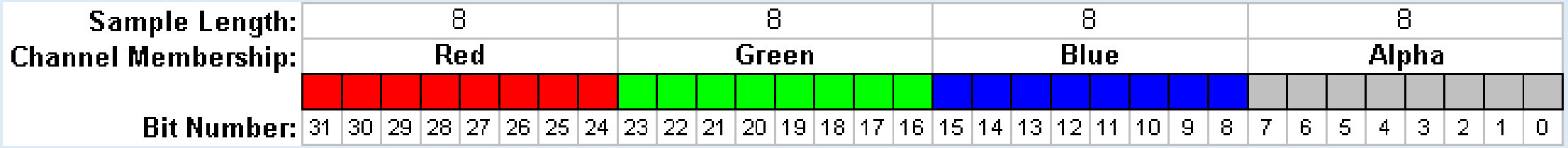
位掩码 Bitmask

- 对一组位模式进行**选择性操作**，即屏蔽掉部分位，只对感兴趣的部分进行修改，这种操作称为**位掩码**（Bitmask）。
- **切换位的状态**
  - 有些情况下，我们需要关闭一部分位的同时打开另一部分。
  - 按位异或位操作 ^ 可以实现精准的控制。

```
  10011101    10010101
^ 00001111    11111111    ~ 10010101
  --------    --------      --------
  10010010    01101010      01101010
```

综合应用：RGBA32 色轮

- RGBA32 值通常使用 8 个十六进制数字表示，每对十六进制数字分别表示红色、绿色、蓝色和 Alpha 通道的值。
- 通道可以使用一个 32 位无符号整数表示，位的排列方式是红色通道位于最高的 8 位，然后是绿色通道、蓝色通道，最低 8 位是 Alpha 通道。



问题？