话题——

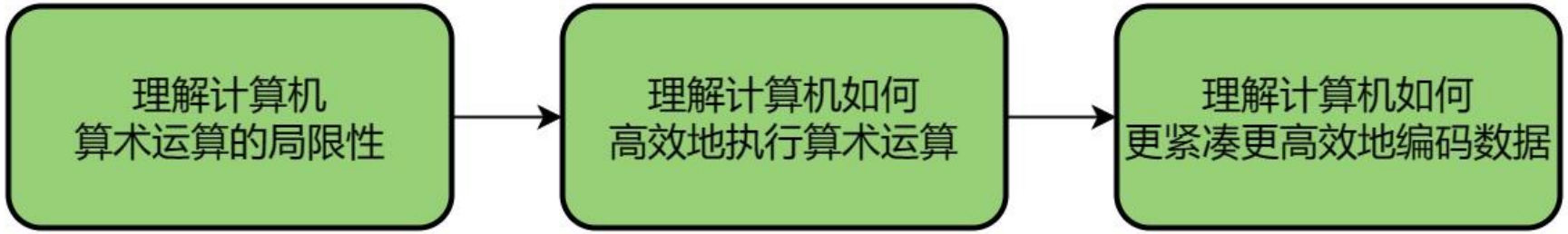
数据的表示，Pt 2

- 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

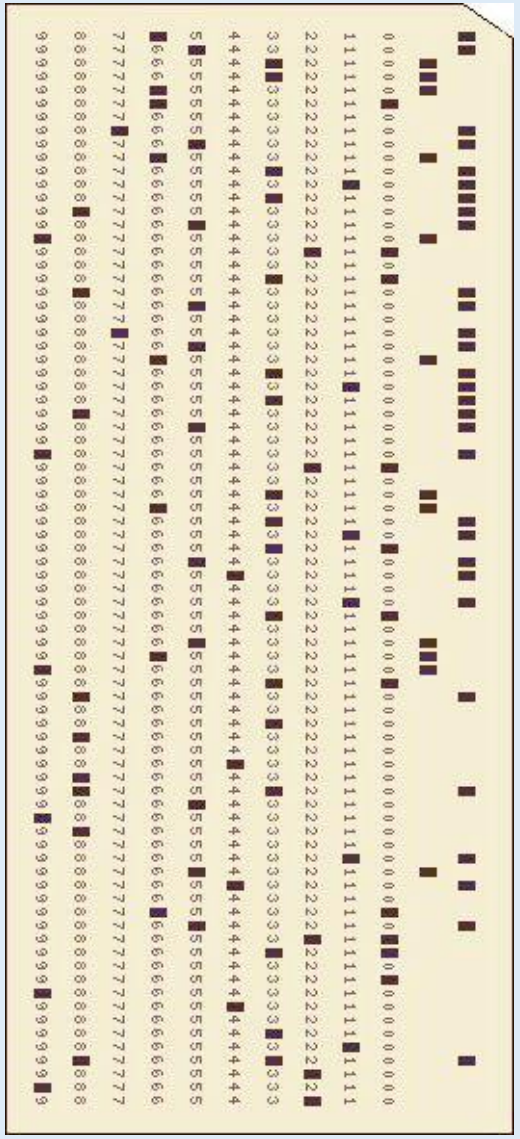
- 练习数字的表示、位操作、位掩码操作
- 阅读分析操作位和整型的 C 代码
- 进一步练习 Linux 环境的开发流程

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

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}

字 Word

- 32 位和 64 位代表计算机的字长（word size），表示指针的大小。
- 虚拟地址是以字来编码的，所以字长决定了计算机虚拟地址空间的大小。

Types	Bytes（32 Bit)	Bytes（64 Bit)
char	1	1
short	2	2
int	4	4
long	**4**	**8**
float	4	4
double	8	8
char *	**4**	**8**

补充 数据类型宽度

sizeof 可以将数据类型或表达式作为参数，并返回该类型或表达式类型的字节数。

语法：`sizeof(type)` 或 `sizeof expression`

示例：`sizeof.c`

参考：https://en.cppreference.com/w/c/language/sizeof

补充 printf 输出有符号/无符号整型

查询 **man 3 printf** 可以了解一些其他的输出占位符，针对整型输出形式有如下一些补充。

- **%d** - 有符号整型按十进制形式输出
- **%u** - 无符号整型按十进制形式输出
- **%x** - 无符号整型按十六进制形式输出

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

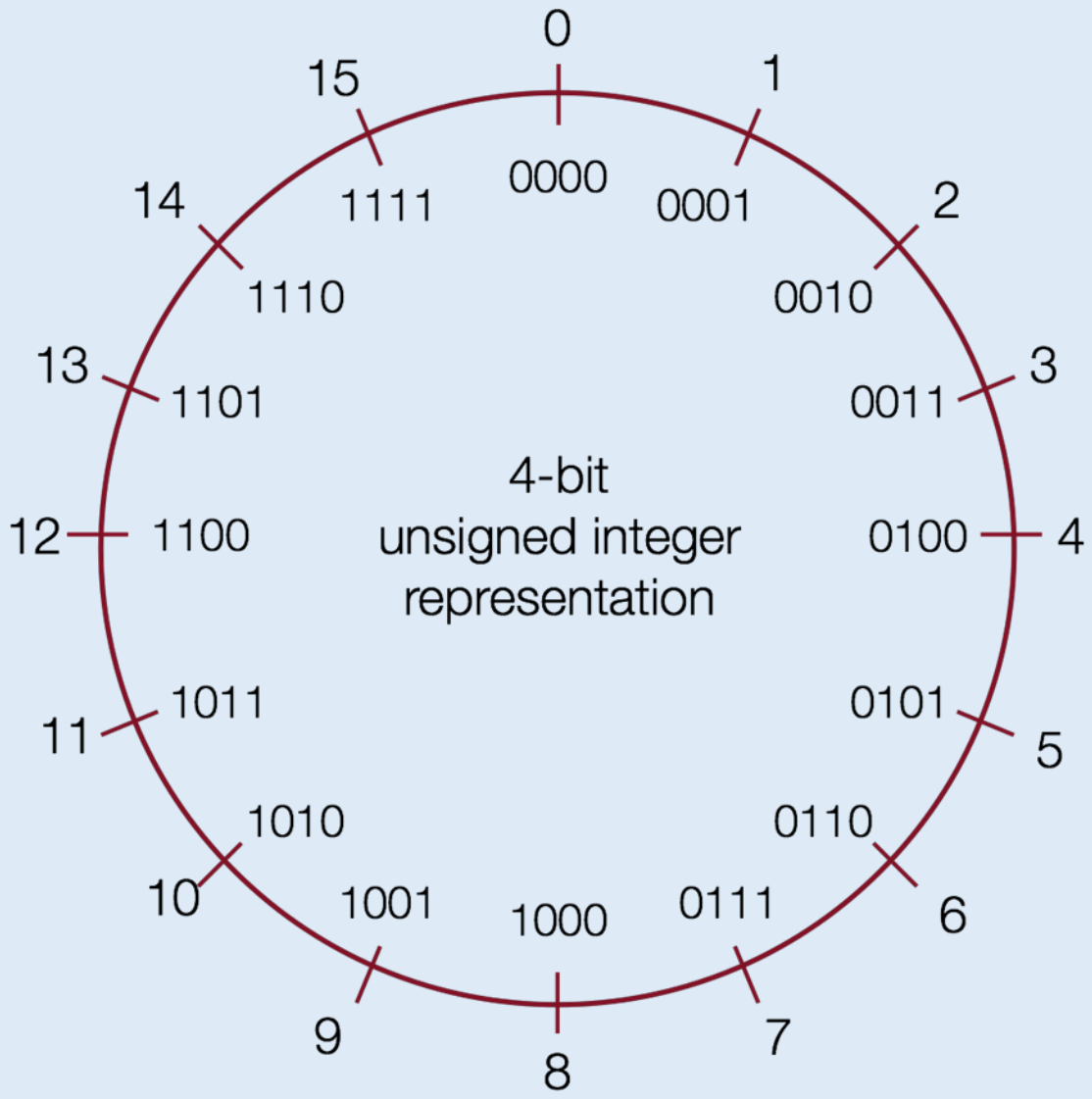
0b10100101

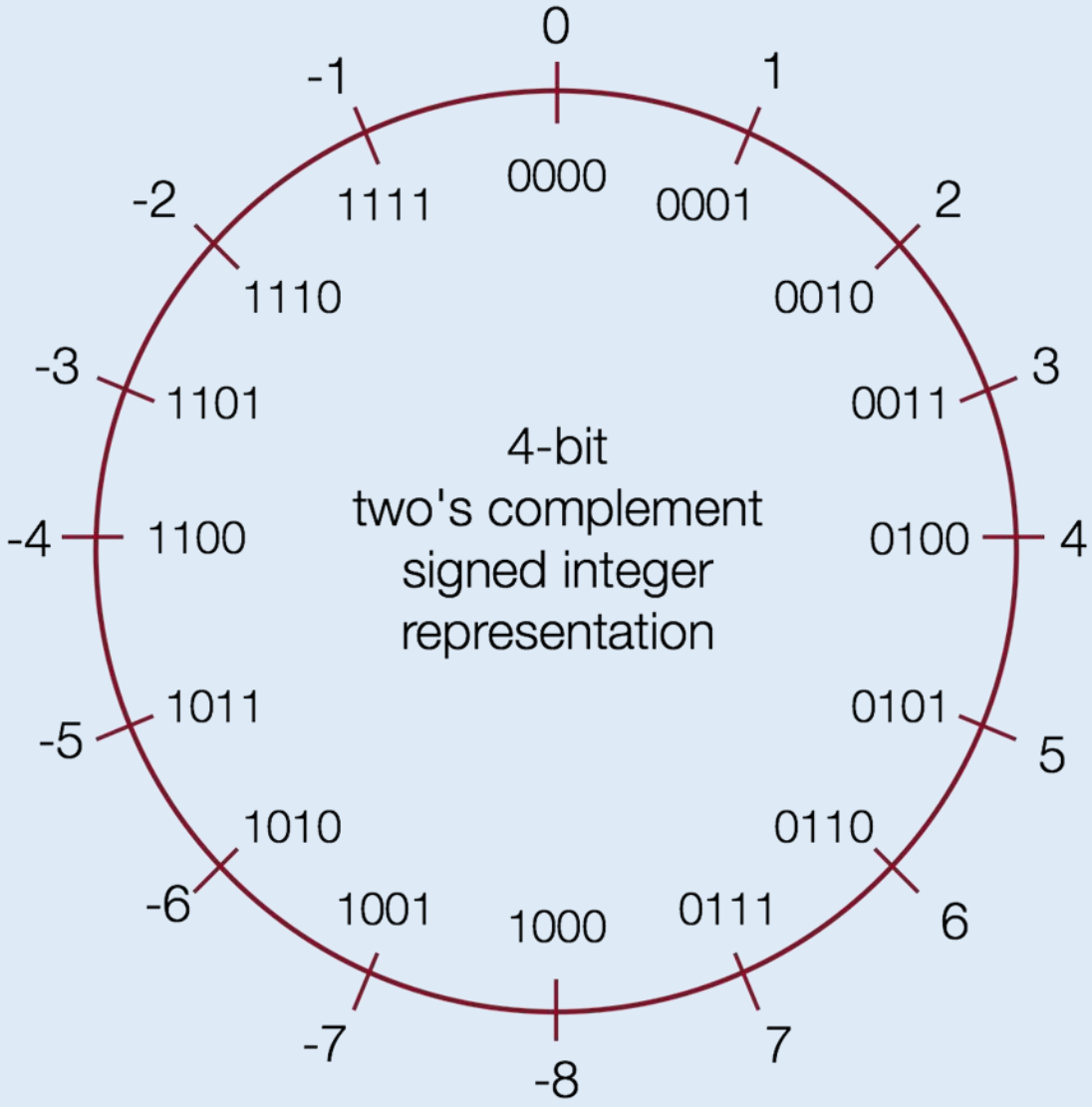
二进制
- 计算机使用的格式
- 可读性差

0xa5

十六进制
- 容易转换到二进制位模式
- 可读性好

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

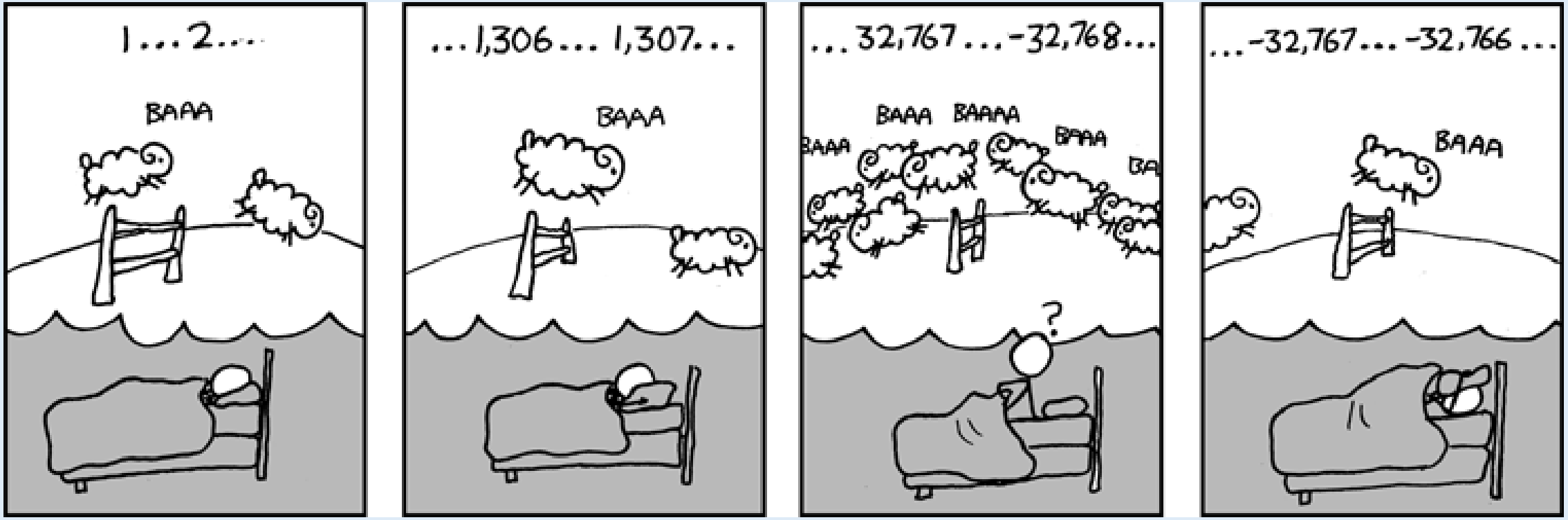
问题？

今日话题

- **Overflow**
- Casting
- Expanding
- Truncating

推荐阅读：

- CSAPP, Ch2


1...2....
BAAA
...1,306... 1,307...
BAAA
...32,767...-32,768...
BAAA
BAAAA
BAAA
BAAA
?
...-32,767...-32,766...
BAAA

Comair/Delta 航空公司因 16 位指针溢出而崩溃

Comair/Delta 航空公司的一款软件隐藏了一个限制，每个月人员分配不得超过 32768 次（类型 short 最大值）。下述代码模拟了这个 bug 的发生过程：

```c
// airline.c

int main(int argc, char *argv[]) {

  short crew_changes = 0;
  // Simulate accumulating crew changes over time

  for (int i = 0; i < 31; i++) {
    // Pretend there are a certain number of crew changes per day

    crew_changes += 1200;

    printf("Crew changes by end of day %d: %d\n", i + 1, crew_changes);

  }

}
```

参考：https://arstechnica.com/uncategorized/2004/12/4490-2/

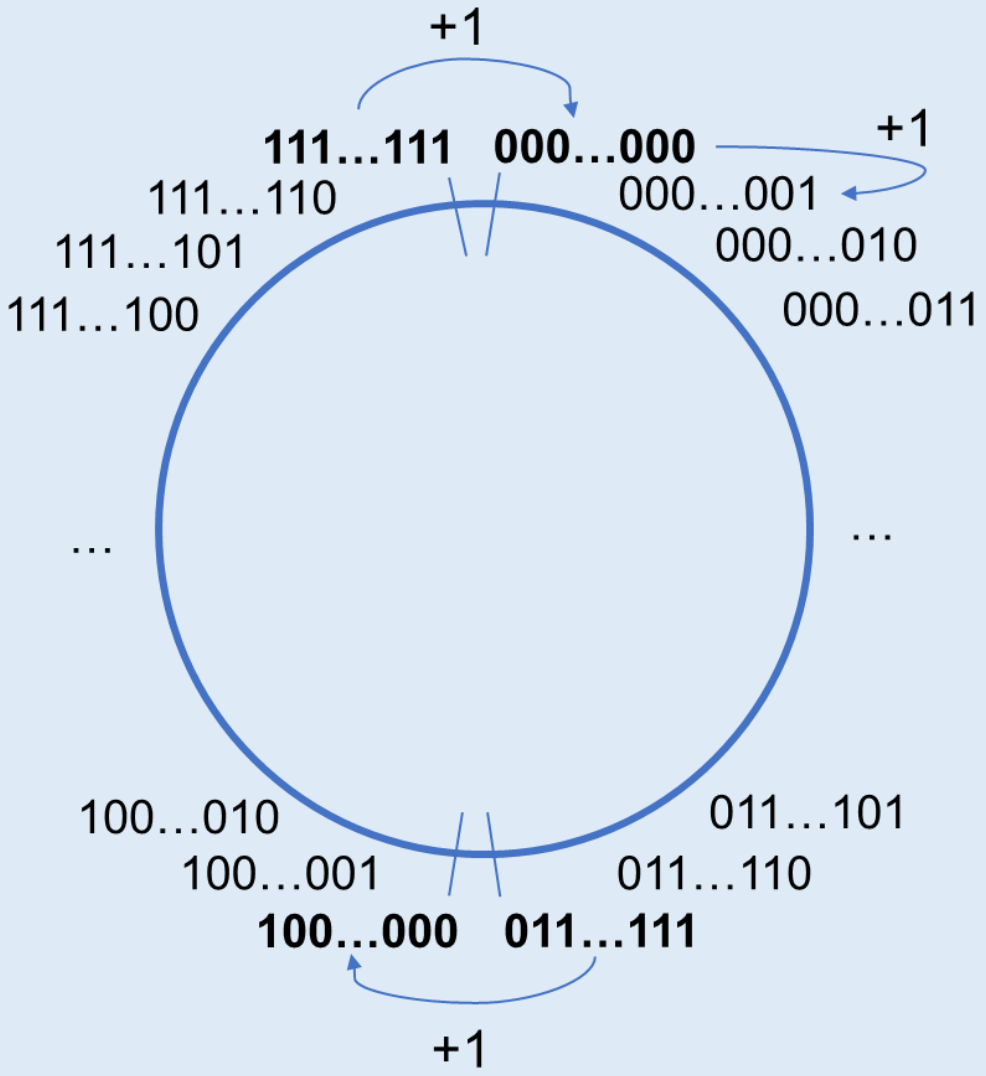
整型溢出

所谓“整型溢出”，就是这样一种现象，即

- 最大的位模式表示 0b1111 加一个较小的数值时，比如 0b1 或 0b10，每个位都需要作进位操作。对于有限的位，这将造成最后的一次进位丢失，无法表示出来，最终变为较小的位模式。
  - `0b1111 + 0b1 = 0b0000`
- 最小的位模式表示 0b0000 减一个较小的数值时，比如 0b1 或 0b10，每个位都需要作借位操作。同样对于有限的位，最后一次借位不会表示出来，最终结果会变为较大的位模式。
  - `0b0000 - 0b1 = 0b1111`

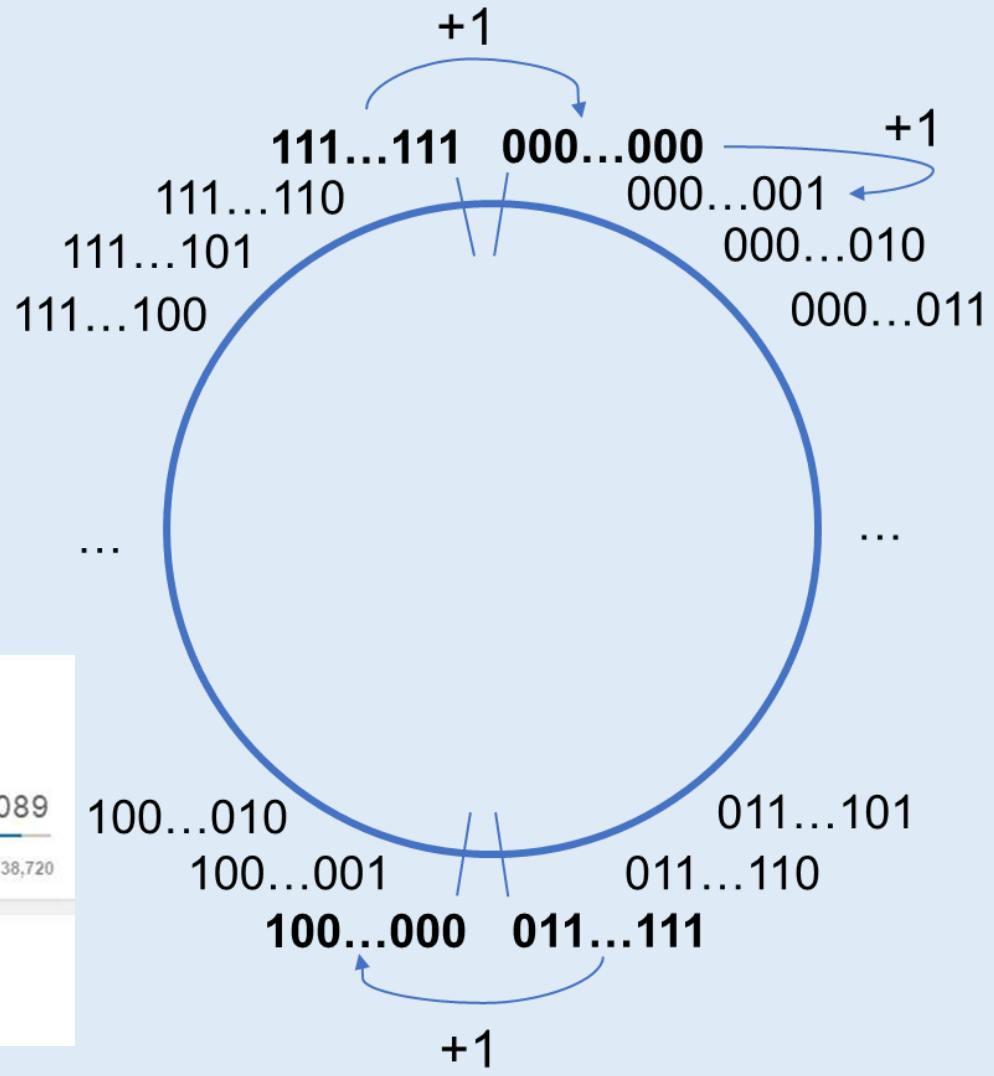
溢出的位置

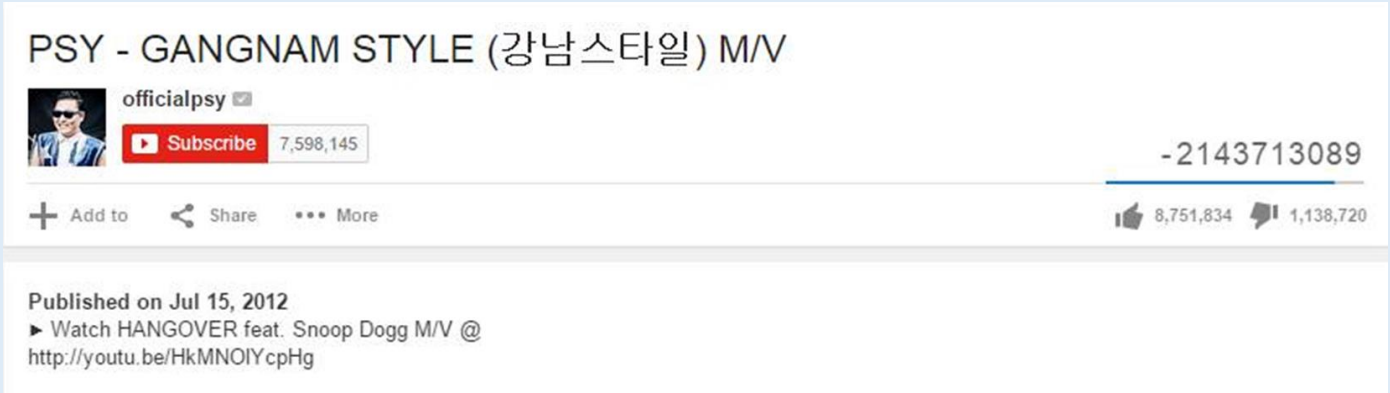
- 对于有符号整型，溢出发生在轮盘的下方
- 对于无符号整型，溢出发生在轮盘的上方



溢出的位置

- 对于有符号整型，溢出发生在轮盘的下方
- 对于无符号整型，溢出发生在轮盘的上方





C 标准保证范围

- C标准要求数据类型至少有这样的值范围。

C data type	Minimum	Maximum
[signed] char	−127	127
unsigned char	0	255
short	−32,767	32,767
unsigned short	0	65,535
int	−32,767	32,767
unsigned	0	65,535
long	−2,147,483,647	2,147,483,647
unsigned long	0	4,294,967,295
int32_t	−2,147,483,648	2,147,483,647
uint32_t	0	4,294,967,295
int64_t	−9,223,372,036,854,775,808	9,223,372,036,854,775,807
uint64_t	0	18,446,744,073,709,551,615

Figure 2.11 **Guaranteed ranges for C integral data types.** The C standards require that the data types have at least these ranges of values.

C 整型典型范围

C data type	Minimum	Maximum
[signed] char	−128	127
unsigned char	0	255
short	−32,768	32,767
unsigned short	0	65,535
int	−2,147,483,648	2,147,483,647
unsigned	0	4,294,967,295
long	−2,147,483,648	2,147,483,647
unsigned long	0	4,294,967,295
int32_t	−2,147,483,648	2,147,483,647
uint32_t	0	4,294,967,295
int64_t	−9,223,372,036,854,775,808	9,223,372,036,854,775,807
uint64_t	0	18,446,744,073,709,551,615

Figure 2.9 Typical ranges for C integral data types for 32-bit programs.

C data type	Minimum	Maximum
[signed] char	−128	127
unsigned char	0	255
short	−32,768	32,767
unsigned short	0	65,535
int	−2,147,483,648	2,147,483,647
unsigned	0	4,294,967,295
long	−9,223,372,036,854,775,808	9,223,372,036,854,775,807
unsigned long	0	18,446,744,073,709,551,615
int32_t	−2,147,483,648	2,147,483,647
uint32_t	0	4,294,967,295
int64_t	−9,223,372,036,854,775,808	9,223,372,036,854,775,807
uint64_t	0	18,446,744,073,709,551,615

Figure 2.10 Typical ranges for C integral data types for 64-bit programs.

查询整型范围

通过导入头文件 `#include <limits.h>` 我们可以使用预先定义好的宏来表示这些最大最小值。

- INT_MIN
- INT_MAX
- UINT_MAX
- LONG_MIN
- LONG_MAX
- ……

今日话题

- Overflow
- **Casting**
- Expanding
- Truncating

推荐阅读：

- CSAPP, Ch2

CASTING

对于下述代码，当我们执行后输出结果是

```
int value = -12345;

unsigned uvalue = value;

printf("v = %d, uv = %u\n", value, uvalue);
```

CASTING

- 当发生类型转换的时候，字节层面的位模式会发生什么呢？
- 答案是

位模式并不会作任何变化。

- 这也就意味着根据类型的不同，同样的位模式将被解释为不同的数值。

CASTING

对于下述代码，当我们执行后输出结果是

```
int value = -12345;
unsigned uvalue = value;
printf("v = %d, uv = %u\n", value, uvalue);
```

```
v = -12345, uv = 4294954951
```

- 有符号整型 -12345 的位模式表示是

0b11111111111111111100111111000111

- 当以无符号整型解释时，这将是一个很大的正数。

C 语言类型转换

- 从有符号整型转向无符号整型，可以在赋值（=）过程中自动完成，我们称之为**隐式类型转换**
- 还可以使用 (typename)value 的语法，进行**强制类型转换**

```
int v = -12345;

...(unsigned int)v...
```

- C 语言中整型字面量默认是是 int 类型，可以通过添加**后缀**转换类型：

```
-12345U - unsigned int

    32L - long

  120UL - unsigned long
```

C 语言比较中的类型转换

- 当对有符号整型和无符号整型进行比较时，C 语言会将有符号整型**升级**（promote）为无符号整型，然后再执行比较操作。

Expression	Comparison Type?	Evaluates To?	Mathematically correct?
0 == 0U	Unsigned	true	yes
-1 < 0			
-1 < 0U			
2147483647 > -2147483648			
2147483647U > -2147483648			
-1 > -2			
(unsigned)-1 > -2			

关键思想 CASTING

相同字长的有符号和无符号数之间的转换规则：

数值可能会变，但是位模式保持不变。

GDB print

- GDB print 几乎可以计算任何表达式，对于算术运算以及在十六进制、十进制和二进制之间转换值非常有用。可以使用 p/format 和不同的格式代码来实现此目的。

```
(gdb) p -12345
$1 = -12345
(gdb) p/u -12345
$2 = 4294954951
(gdb) p/t -12345
$3 = 11111111111111111100111111000111
(gdb) p/x -12345
$4 = 0xffffcfc7
(gdb)
```

格式说明：

- x 代表十六进制
- t 表示二进制
- c 用于字符
- u 表示无符号十进制
- d 表示有符号十进制

```
$ gdb ./casting
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
Reading symbols from ./casting...
(gdb) break main
Breakpoint 1 at 0x1158: file casting.c, line 10.
(gdb) run
Starting program: /root/starter/lectures/lect3/casting
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, main (argc=1, argv=0x7fffffffe4c8) at casting.c:10
10          printf("int -> unsigned int\n");
(gdb) next
int -> unsigned int
11          int value = -12345;
(gdb) next
12          unsigned int uvalue = value;
(gdb) next
13          printf("value = %d, uvalue = %u\n", value, uvalue);
(gdb) p/t value
$1 = 11111111111111111100111111000111
(gdb) p/t uvalue
$2 = 11111111111111111100111111000111
(gdb) q
```

今日话题

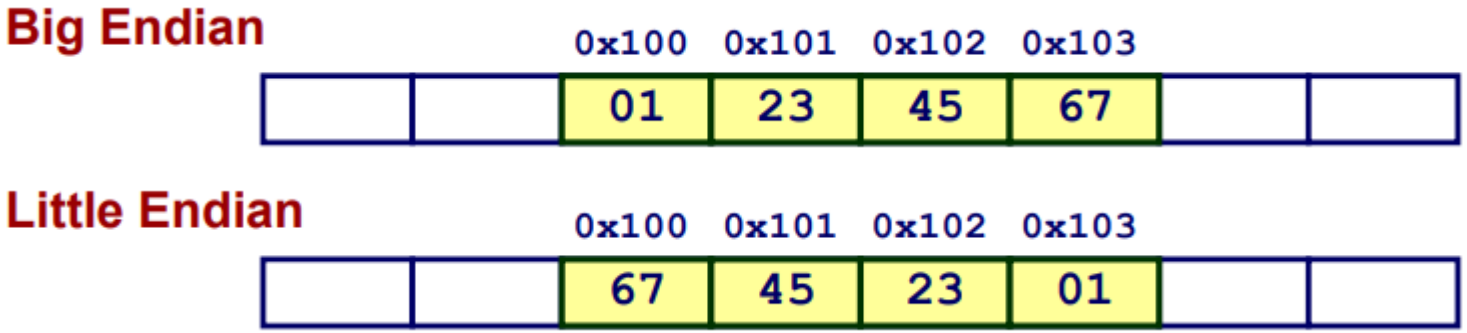
- Overflow
- Casting
- **Expanding**
- Truncating

推荐阅读：

- CSAPP, Ch2

补充 大端小端

4 个字节表示的值 **x = 0x01234567** 在大端和小端系统中的字节分布如下



x 的地址都是 0x100

大端：最低有效字节在高地址，常见于 Sun、Internet 系统中

小端：最低有效字节在低地址，常见于 x86、ARM、Linux 系统中

EXPANDING

- 从较小长度的类型转换到较大长度的类型，称为**扩展**（Expanding）
- **零扩展**（Zero Extension）：将一个**无符号整型**提升到较大字长的无符号整型时，只需要简单地在开头补充 0，这种运算被称为零扩展。

```
unsigned short us = 4;  //                         0000 0000 0000 0100
unsigned int ui = us;   // 0000 0000 0000 0000 0000 0000 0000 0100
```

- **符号扩展**（Sign Extension）：将一个**有符号整型**提升到较大字长的有符号整型时，则需要在开头补充最高有效位的值（0 或 1），这种运算被称为符号扩展。

```
short s = 4;  //                         0000 0000 0000 0100
int i = s;    // 0000 0000 0000 0000 0000 0000 0000 0100
short s = -4; //                         1111 1111 1111 1100
int i = s;    // 1111 1111 1111 1111 1111 1111 1111 1100
```

关键思想 EXPANDING

- 从较小的数据类型转向较大的数据类型**不会造成数值的改变**。
- 对不同字长大小的数据类型作比较操作（<、>、<=、>=），会将较小字长的数据**升级**（promote）到较大字长，然后再进行比较。
- 根据不同数据类型，有**零扩展**和**符号扩展**两种。

今日话题

- Overflow
- Casting
- Expanding
- **Truncating**

推荐阅读：

- CSAPP, Ch2

TRUNCATING

- 从较大长度的类型转换到较小长度的类型，称为**截断**（Truncating）；在 C 语言中，截断操作会将多余的字节将会直接忽略。
- 无符号整型

```
unsigned int uv = 128000; // 0000 0000 0000 0001 1111 0100 0000 0000 (128000)
unsigned short us = uv;   //                      1111 0100 0000 0000 (62464)
```

- 有符号整型

```
int x = 53191;  // 0000 0000 0000 0000 1100 1111 1100 0111 (53191)
short sx = x;   //                    1100 1111 1100 0111 (-12345)


int x = -54321; // 1111 1111 1111 1111 0010 1011 1100 1111 (-54321)
short sx = x;   //                    0010 1011 1100 1111 (11215)
```

关键思想 TRUNCATING

与扩展不同，截断一般会改变原来的数值，仅较小的数可以保持数值不变。

截断 vs 溢出

问题 7

casting.c expanding.c
truncating.c