信息的表示和处理(2)

谢旻晖

2025.9

Outline

- Integers
  - **Representation: unsigned and signed**
  - Conversion, casting
  - Expanding, truncating
  - Addition, negation, multiplication, shifting
  - Summary

Unsigned Representation

- Binary (physical)
  - Bit vector $[x_{w-1}, x_{w-2}, x_{w-3}, \ldots x_0]$
- Binary to Unsigned (logical)

$$B2U(X) = \sum_{i=0}^{w-1} x_i \cdot 2^i$$

例如：N_1=01011，表示无符号数11;

N_2=11011 表示无符号数27;

N_3=00000 表示无符号数0。

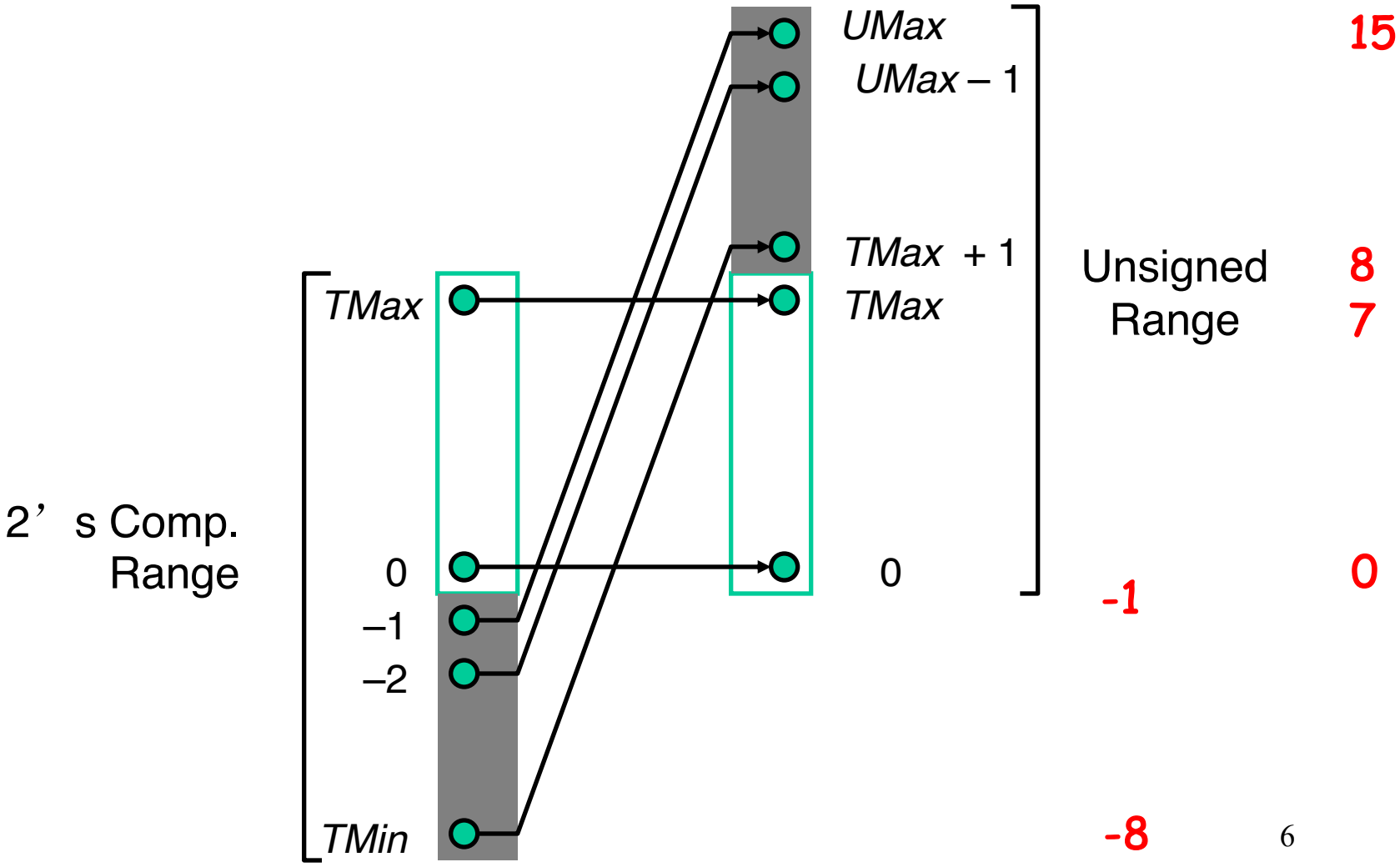
Unsigned Representation

- 如果机器字长为n位，则无符号数的表示范围是：
  $0\sim(2^n-1)$。
- 例如字长为8位，则无符号数的表示范围是0～255。
- 计算机的内存地址就是无符号数的例子。

Signed Representation

- Binary原本的含义与非负整数正好对应
- 负数如何用binary表达？
  - 解决思路：**映射**(mapping)
  - 将负数段映射到一个不使用的非负数段
    - 对有限范围的整数成立
    - 最好不影响正数和0
  - 且最好负数和映射到的整数在某种意义上是"**等价**"的，可以直接进行运算

补码的映射关系


2’s Comp. Range
TMax
0
–1
–2
TMin
UMax
UMax – 1
TMax + 1
TMax
0
Unsigned Range
15
8
7
0
-1
-8

Signed Representation

- 假设时钟停在 8点，而现在正确的时间是6点，这时拨准时钟的方法有两种：
  - 一是将分针倒着旋转2圈（即时钟倒拨2小时），8-2=6（做减法）
  - 另一种方法是将分针正着旋转 10圈（即时钟正拨10小时），8+10=6（mod 12）（做加法）
- 此时： **8-2 = 8+10 (mod 12)**
- 所以有：**-2=10 (mod 12)**，即**-2和10同余**。
- 同余的两个数，具有互补关系，-2和10 对模12互补，也就是-2的补数是10（以12为模）

Signed Representation

补码表示法

- 只要确定了"模"，就可找到一个与负数等价的正数（该正数即为负数的补数）来代替此负数，而这个正数可以用模加上负数本身求得，这样就可把减法运算用加法实现了。
- 例： 9-5=9+(12-5)=9+7=4 (mod 12)

  65-25=65+(-25)=65+(100-25)=65+75=40 (mod 100)

Two's Complement

- Binary (physical)
  - Bit vector $[x_{w-1}, x_{w-2}, x_{w-3}, \ldots x_0]$
- Binary to Signed (logical)

w位，有2^w个状态，
所以模就是2^w；
增加或减少2^w，值不变

$$B2T(X) = -x_{w-1} \cdot 2^{w-1} + \sum_{i=0}^{w-2} x_i \cdot 2^i$$

Sign Bit

- 2's complement
  - “模”就是2^w
  - 因为binary形式的值是$2^{w-1}+\ldots$，实际值是$-2^{w-1}+\ldots$，二者相差2^w

From a Number to Two's Complement

- 真值-5
  - -5+16 = 11
-
  - y=5
  - 0101 (binary for 5)
  - 1010 (after complement)
  - 1011 (add 1)
  - 补码x为1011

真值 ←→ 补码：
1）模-真值绝对值/补码
（如10000-0101）
2）每一位都取反，最后加1

From Two's Complement to Number

- 补码x=1011
  - 1011 (two's complement binary)
  - 0100 (after complement)
  - 0101 (add 1)
  - Y=0101, 真值-y=-5

Two's Complement Encoding Examples

Binary/Hexadecimal Representation for -12345

Binary: 0011 0000 0011 1001 (12345)

Hex: 3 0 3 9

Binary: 1100 1111 1100 0110 (after complement)

Hex: C F C 6

Binary: 1100 1111 1100 0111 (add 1)

Hex: C F C 7

Numeric Range

- Unsigned Values
  - Umin=0
  - Umax=2^w-1
- Two's Complement Values
  - Tmin = -2^{w-1}
  - Tmax = $2^{w-1}-1$

E.g., w=4
0000 ~ 0111: 0~7
1000 ~ 1111: -8~-1
-(0111+1) -(0000+1)

Values for Different Word Sizes

	W			
	8	**16**	**32**	**64**
UMax	255	65,535	4,294,967,295	18,446,744,073,709,551,615
TMax	127	32,767	2,147,483,647	9,223,372,036,854,775,807
TMin	-128	-32,768	-2,147,483,648	-9,223,372,036,854,775,808

- Observations
  - $|TMin| = TMax + 1$
    - Asymmetric range
  - $UMax = 2 * TMax + 1$

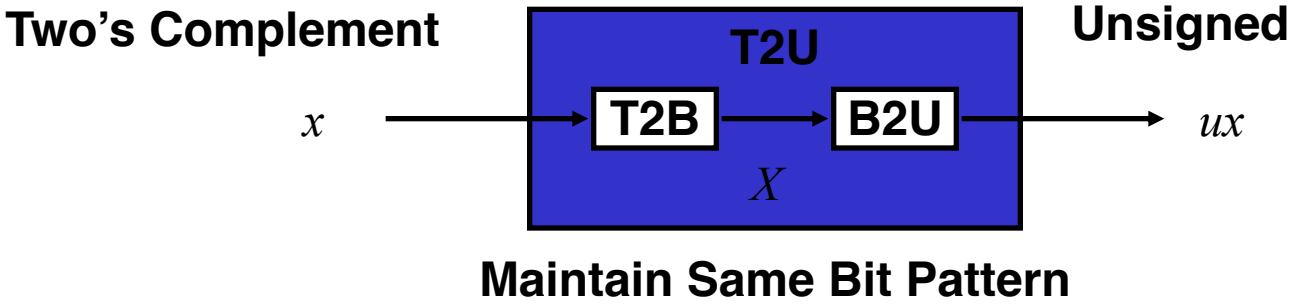
- **C Programming**
  - #include <limits.h>
  - Declares constants, e.g.,
    - ULONG_MAX
    - LONG_MAX
    - LONG_MIN
  - Values platform specific

Alternative representations of signed numbers

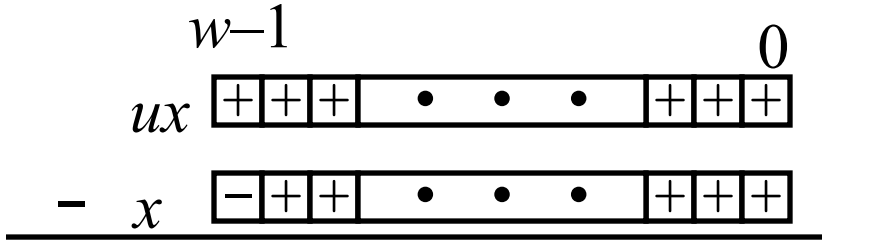
- Ones' Complement (反码)
  - The most significant bit has weight $2^{w-1}-1$
- Sign-Magnitude (原码)
  - The most significant bit is a sign bit
    - that determines whether the remaining bits should be given negative or positive weight

X	B2U(*X*)	B2T(*X*)
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	–8
1001	9	–7
1010	10	–6
1011	11	–5
1100	12	–4
1101	13	–3
1110	14	–2
1111	15	–1

Relation Between 2's Comp. & Unsigned



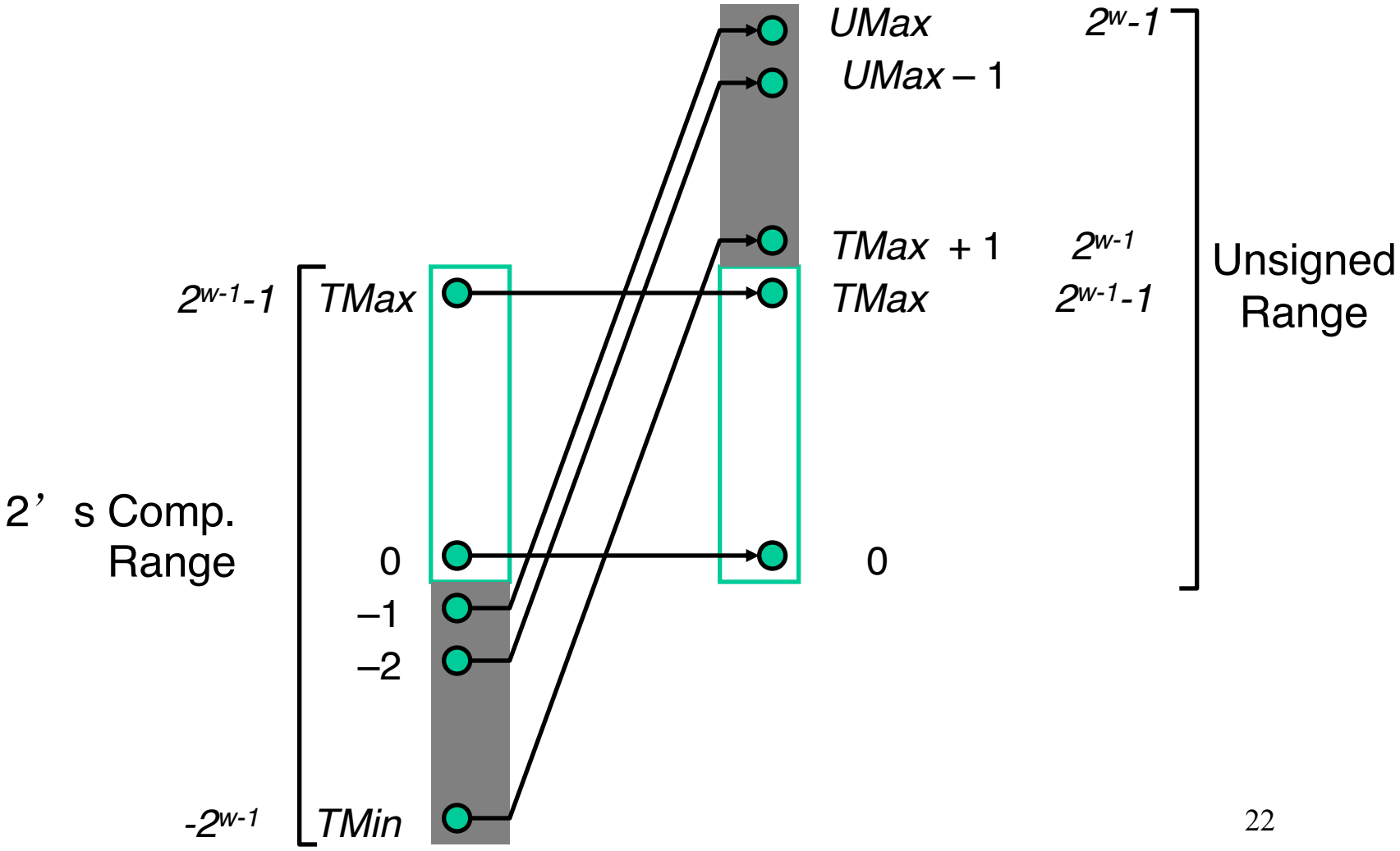
二进制形式不变，
理解方式不同
（负数变正数）



$$+2^{w-1} - -2^{w-1} = 2*2^{w-1} = 2^{w}$$

$$ux = \begin{cases} x & x \geq 0 \\ x + 2^{w} & x < 0 \end{cases}$$

Conversion between two Representations



课堂练习

- 求+0.1101、-0.1101的补码
- 写出下列数字的补码（short类型）【16进制】
  - -178
- char x =10110111b
  - 如果是补码，则真值是多少？

课堂测试

- 写出下列数字的补码（short类型）【16进制】
  - -39
  - -26
- 下面给出了一组short类型的变量在计算机内的编码（补码），请写出他们的真值(16进制形式)。
  - FF3C
  - FF86

Outline

- Integers
  - Representation: unsigned and signed
  - **Conversion, casting**
  - Expanding, truncating
  - Addition, negation, multiplication, shifting
  - Summary

Integral data type in C

- Signed type (for integer numbers)
  - char, short [int], int, long [int]
- Unsigned type (for nonnegative numbers)
  - unsigned char, unsigned short [int],
    unsigned [int], unsigned long [int]
- Java 没有无符号类型
  - Byte: signed char

Integral Data Types

- C supports a variety of integral data types
  - Represent a finite range of integers

C declaration	guaranteed		Typical 32-bit	
	minimum	maximum	minimum	maximum
char	-127	127	-128	127
unsigned char	0	255	0	255
short [int]	-32,767	32,767	-32,768	32,767
unsigned short	0	65,535	0	65,535
int	-32,767	32,767	-2,147,483,648	2,147,483,647
unsigned [int]	0	65,535	0	4,294,967,295
long [int]	-2,147,483,647	2,147,483,647	-2,147,483,648	2,147,483,647
unsigned long	0	0	0	4,294,967,295

Casting among Signed and Unsigned in C

- C允许一种类型按照另一种类型的方式来理解
  - Type conversion (implicitly)
  - Type casting (explicitly)

Signed vs. Unsigned in C

- Casting
  - Signed和unsigned类型之间的显式转换
  - 即 U2T 和 T2U
    - int tx, ty;
    - unsigned ux, uy;
    - tx = (int) ux;
    - uy = (unsigned) ty;

Signed vs. Unsigned in C

- Conversion
  - 通过赋值语句或call语言（函数调用）来隐式转换
    - int tx, ty;
    - unsigned ux, uy;
    - tx = ux;
    - uy = ty;

    - int func () {…}
    - unsigned ux = func();

Casting from Signed to Unsigned

```
short int          x =  12345;
unsigned short int ux = (unsigned short) x;
short int          y  = -12345;
unsigned short int uy = (unsigned short) y;
```

- 转换的结果结果
  - 非负值不变
    - *ux* = 12345
  - 负值被转为(更大的)正值
    - *uy* = 53191

Weight	12,345		−12,345		53,191	
	Bit	Value	Bit	Value	Bit	Value
1	1	1	1	1	1	1
2	0	0	1	2	1	2
4	0	0	1	4	1	4
8	1	8	0	0	0	0
16	1	16	0	0	0	0
32	1	32	0	0	0	0
64	0	0	1	64	1	64
128	0	0	1	128	1	128
256	0	0	1	256	1	256
512	0	0	1	512	1	512
1,024	0	0	1	1,024	1	1,024
2,048	0	0	1	2,048	1	2,048
4,096	1	4096	0	0	0	0
8,192	1	8192	0	0	0	0
16,384	0	0	1	16,384	1	16,384
±32,768	0	0	1	−32,768	1	32,768
Total		12,345		−12,345		53,191

back

Unsigned Constants in C

- 默认情况
  - 常量被当作有符号数
- 如果希望是无符号数，在后面加上后缀U
  - 0U, 4294967259U

Casting Convention

- 表达式比较
  - 如果在一个表达式中混用unsigned和signed
    - 会隐式地把有符号数专为无符号数
  - 包括 <, >, ==, <=, >= 等比较符号
  - Examples for *W* = 32
    - TMIN = -2,147,483,648 , TMAX = 2,147,483,647

Casting Convention

Constant1	Constant2	Relation	Evaluation
0	0U	==	unsigned
-1	0	<	signed
-1	0U	>	unsigned
2147483647	-2147483648	>	signed
2147483647U	-2147483648	<	unsigned
-1	-2	>	signed
(unsigned)-1	-2	>	unsigned

Outline

- Integers
  - Representation: unsigned and signed
  - Conversion, casting
  - **Expanding, truncating**
  - Addition, negation, multiplication, shifting
  - Summary

From short to long

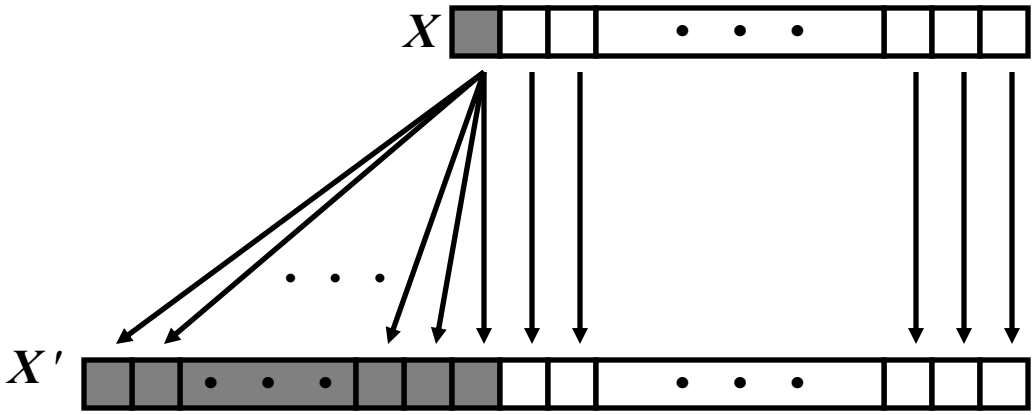
```
short int x =  12345;
int ix = (int) x;
short int y = -12345;
int iy = (int) y;
```

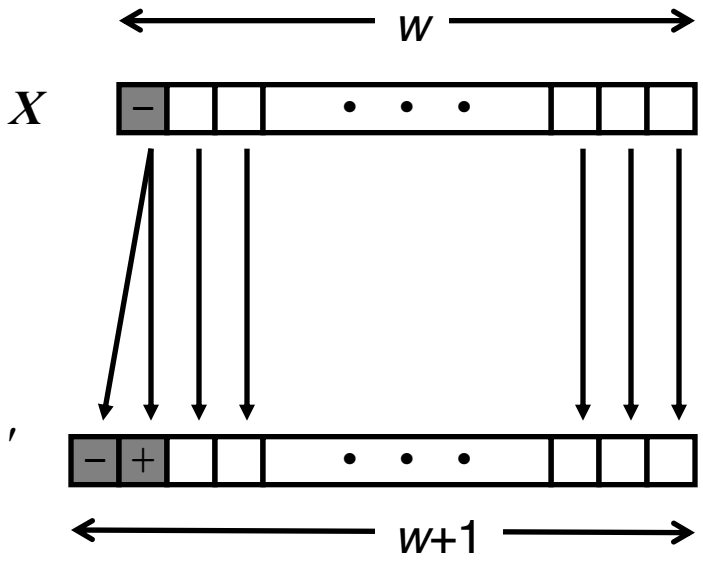
- 需要扩展数据大小

- 在无符号数据之间转换(casting)正常处理即可
- 在有符号数据之间转换(casting)需要特别注意

Expanding the Bit Representation

- Zero extension
  - 在前面添加0
- Sign extension
  - $[x_{w-1}, x_{w-2}, x_{w-3}, \ldots x_0]$





From short to long

```
   short int sx = 12345; : 0x0309
int x = (int) sx;      : 0x00000309
   short int sy =-12345;: 0xcfc7
int y = (int) sy;      : 0xffffcfc7
```

From short to long

```
int fun1(unsigned word) {
        return (int) ((word << 24) >> 24);
}
int fun2(unsigned word) {
        return ((int) word << 24) >> 24;
}
```

w	fun1(w)	fun2(w)
0x00000076	**00000076**	**00000076**
0x87654321	**00000021**	**00000021**
0x000000C9	**000000C9**	**FFFFFFC9**
0xEDCBA987	**00000087**	**FFFFFF87**

练习：描述一下两个函数的行为区别

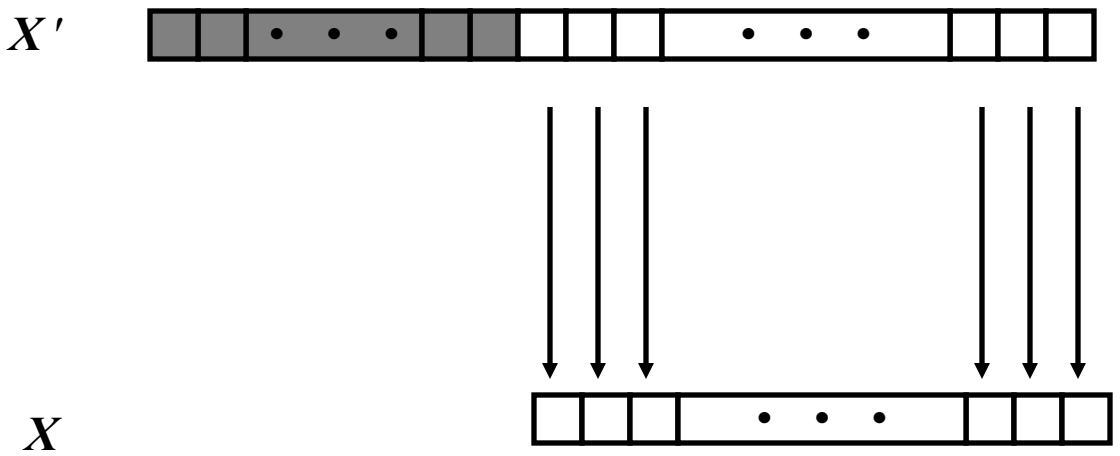
From long to short

```
int         x  = 53191;
short int sx = x;
int         y  = -12345;
Short int sy = y;
```

- 需要截断数据大小
- 从长类型到短类型转换需要特别注意

Truncating Numbers

	Decimal	Hex	Binary
x	53191	00 00 CF C7	00000000 00000000 11001111 11000111
sx	-12345	CF C7	11001111 11000111
y	-12345	FF FF CF C7	11111111 11111111 11001111 11000111
sy	-12345	CF C7	11001111 11000111



Advice on Signed vs. Unsigned

Non-intuitive Features

```
float sum_elements ( float a[], unsigned length )
{
    int i;
    float result = 0;
    for ( i = 0; i <= length – 1; i++)
            result += a[i] ;
}
```

当length=0时，应该返回0.0，但实际会遇到内存错误，为什么？

Advice on Signed vs. Unsigned

```
/* Prototype for library function strlen */
size_t strlen(const char *s); /*size_t is unsigned */

/* Determine whether string s is longer than string t */
/* WARNING: This function is buggy */
int strlonger(char *s, char *t) {
    return strlen(s) - strlen(t) > 0;
}
```

Advice on Signed vs. Unsigned

```
1 /*
2 * Illustration of code vulnerability similar to that found in
3 * FreeBSD's implementation of getpeername (…)
4 * */
5
6 /* Declaration of library function memcpy */
7 void *memcpy(void *dest, void *src, size_t n);
8
```

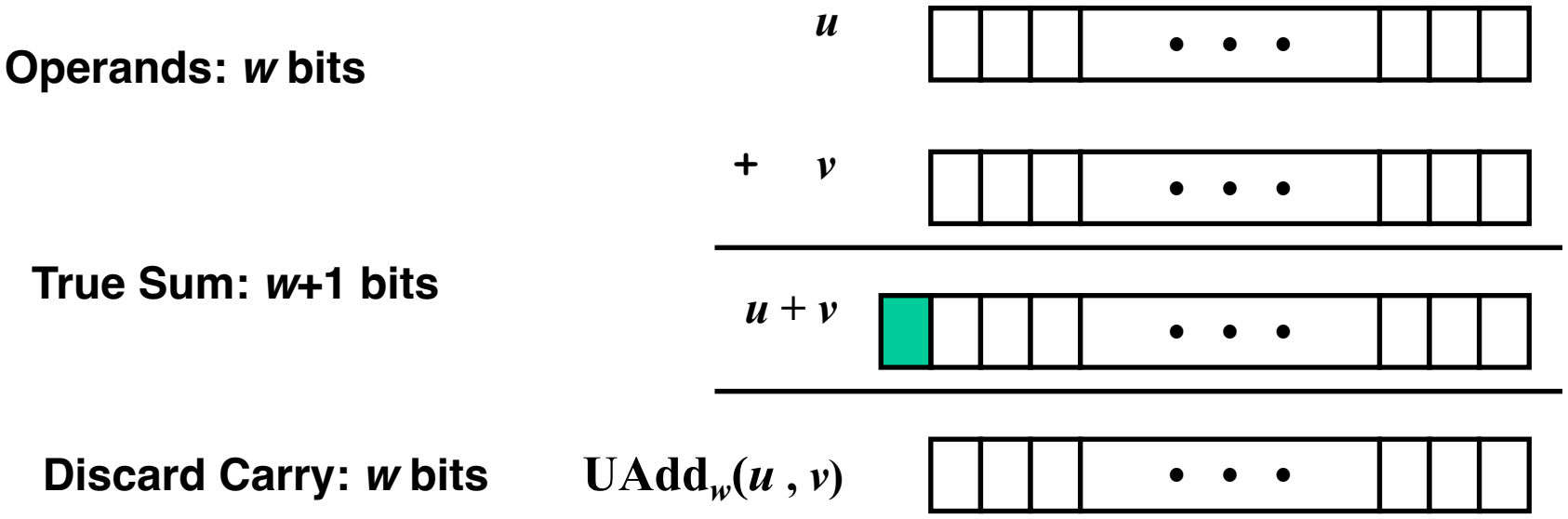
Advice on Signed vs. Unsigned

```
9 /* Kernel memory region holding user-accessible data */
10 #define KSIZE 1024
11 char kbuf[KSIZE];
12
13 /* Copy at most maxlen bytes from kernel region to user buffer */
14 int copy_from_kernel (void *user_dest, int maxlen) {
15       /* Byte count len is minimum of buffer size and maxlen */
16       int len = KSIZE < maxlen ? KSIZE : maxlen;
17       memcpy(user_dest, kbuf, len); // 如果maxlen是个负值
18       return len;
19 }
```

Outline

- Integers
  - Representation: unsigned and signed
  - Conversion, casting
  - Expanding, truncating
  - **Addition, negation, multiplication, shifting**
  - Summary

Unsigned Addition



Unsigned Addition

- Standard Addition Function
  - Ignores carry output
- Implements Modular Arithmetic
  - $s = UAdd_w(u, v) = (u + v) \bmod 2^w$

$$UAdd_w(u,v) = \begin{cases} u+v & u+v < 2^w \\ u+v-2^w & u+v \geq 2^w \end{cases}$$

Unsigned Addition

Practice Problem 2.27

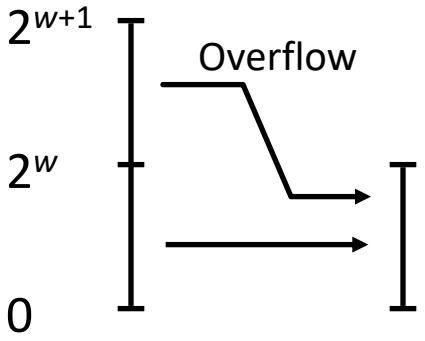
Write a function with the following prototype:

```
/* Determine whether arguments can be added without
   overflow */
int uadd_ok(unsigned x, unsigned y);
```

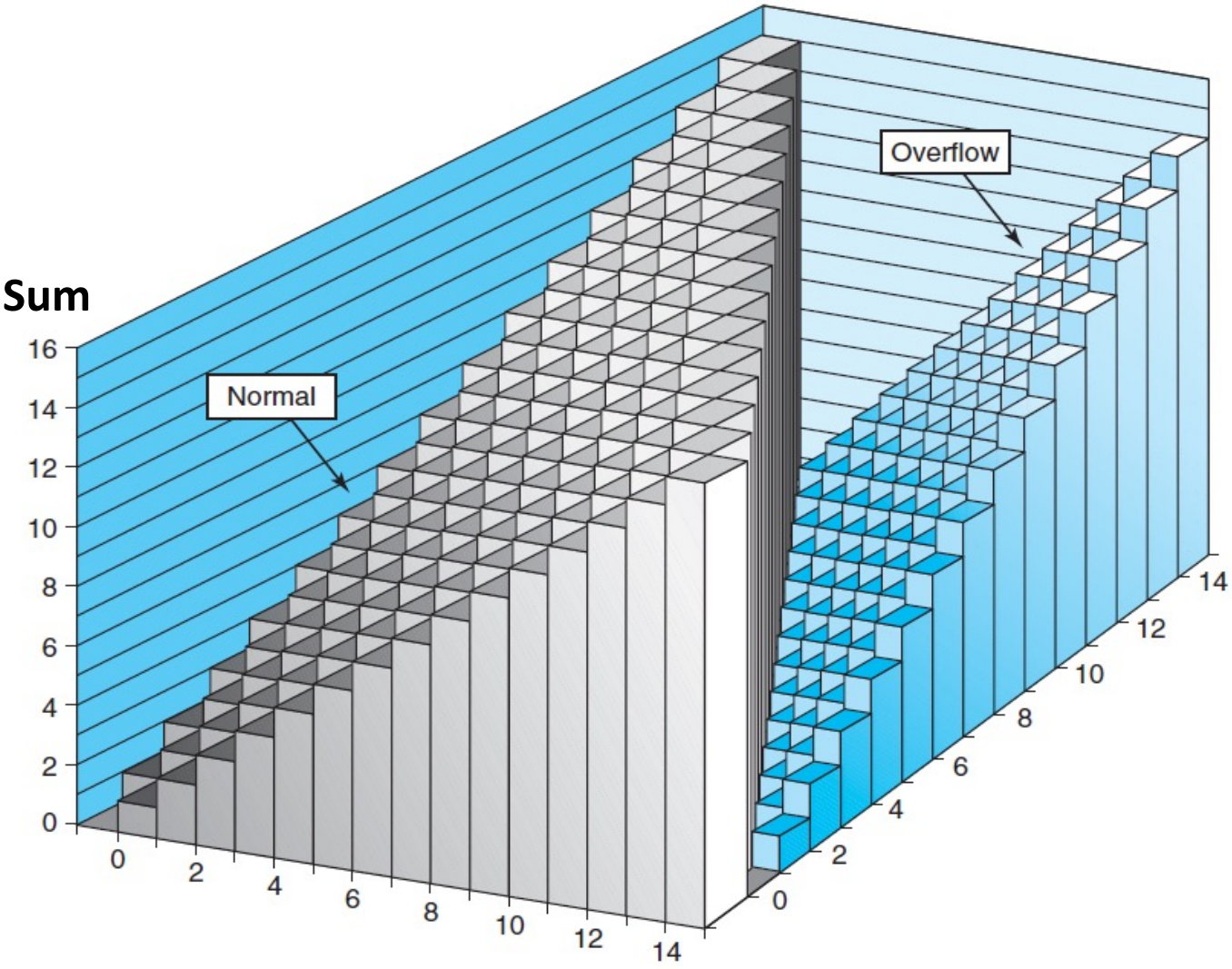
This function should return 1 if arguments x and y can be added without causing overflow

Overflow iff (X+Y) < X

Unsigned Addition



Modular Sum



Unsigned Addition Forms an Abelian Group (阿贝尔群)

- 加法结果是封闭的（还在同样的数据范围内）
  - $0 \le UAdd_w(u, v) \le 2^w - 1$
- 交换律
  - $UAdd_w(t, UAdd_w(u,v)) = UAdd_w(UAdd_w(t, u), v)$
- 有一个单位元0
  - $UAdd_w(u, 0) = u$

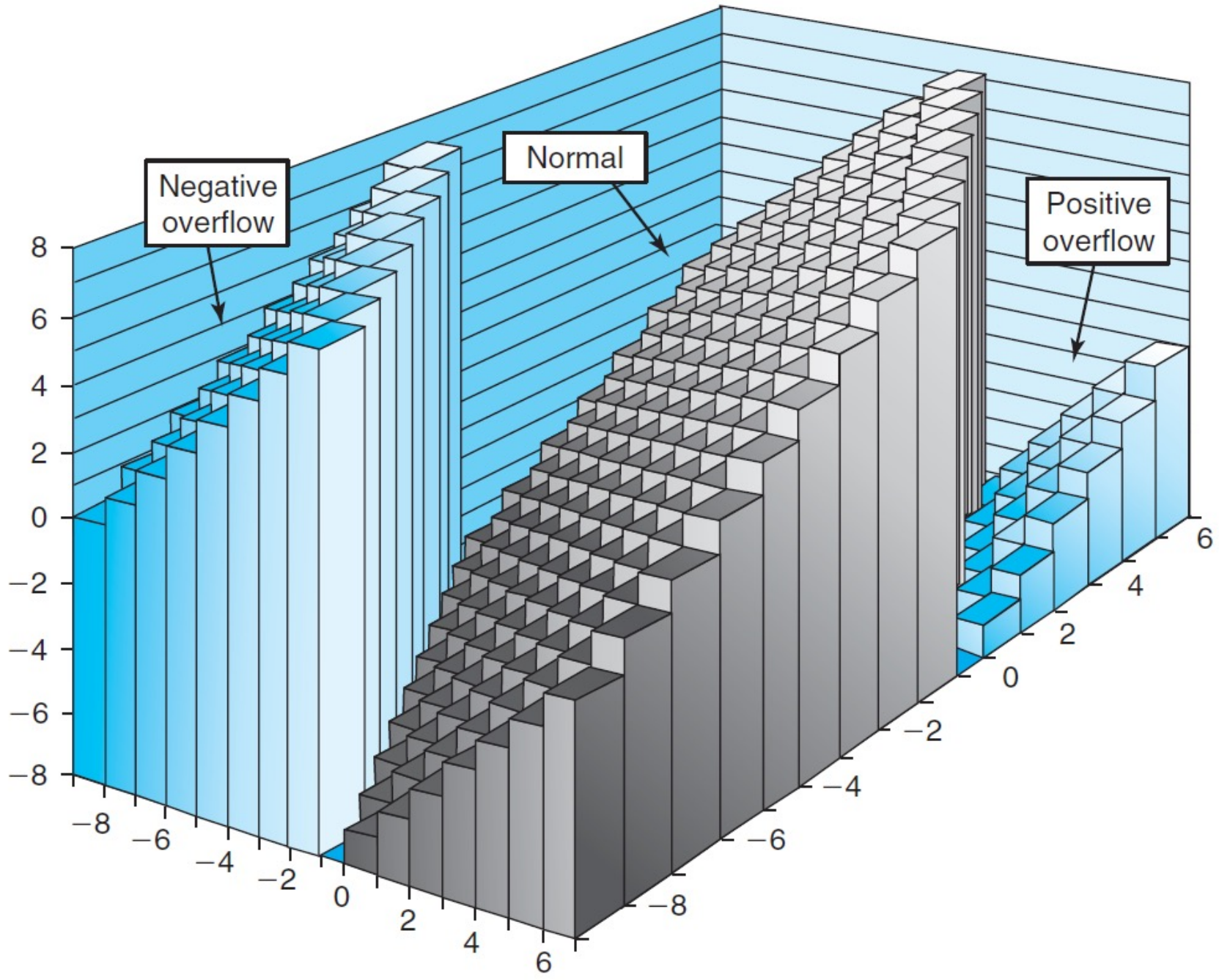
Unsigned Addition Forms an Abelian Group

- 每个元素都有一个加法逆元
  - Let $\text{UComp}_w(u) = 2^w - u$
  - $\text{UAdd}w(u, \text{UComp}_w(u)) = 0$
- 交换律
  - $\text{UAdd}_w(u, v) = \text{UAdd}_w(v, u)$

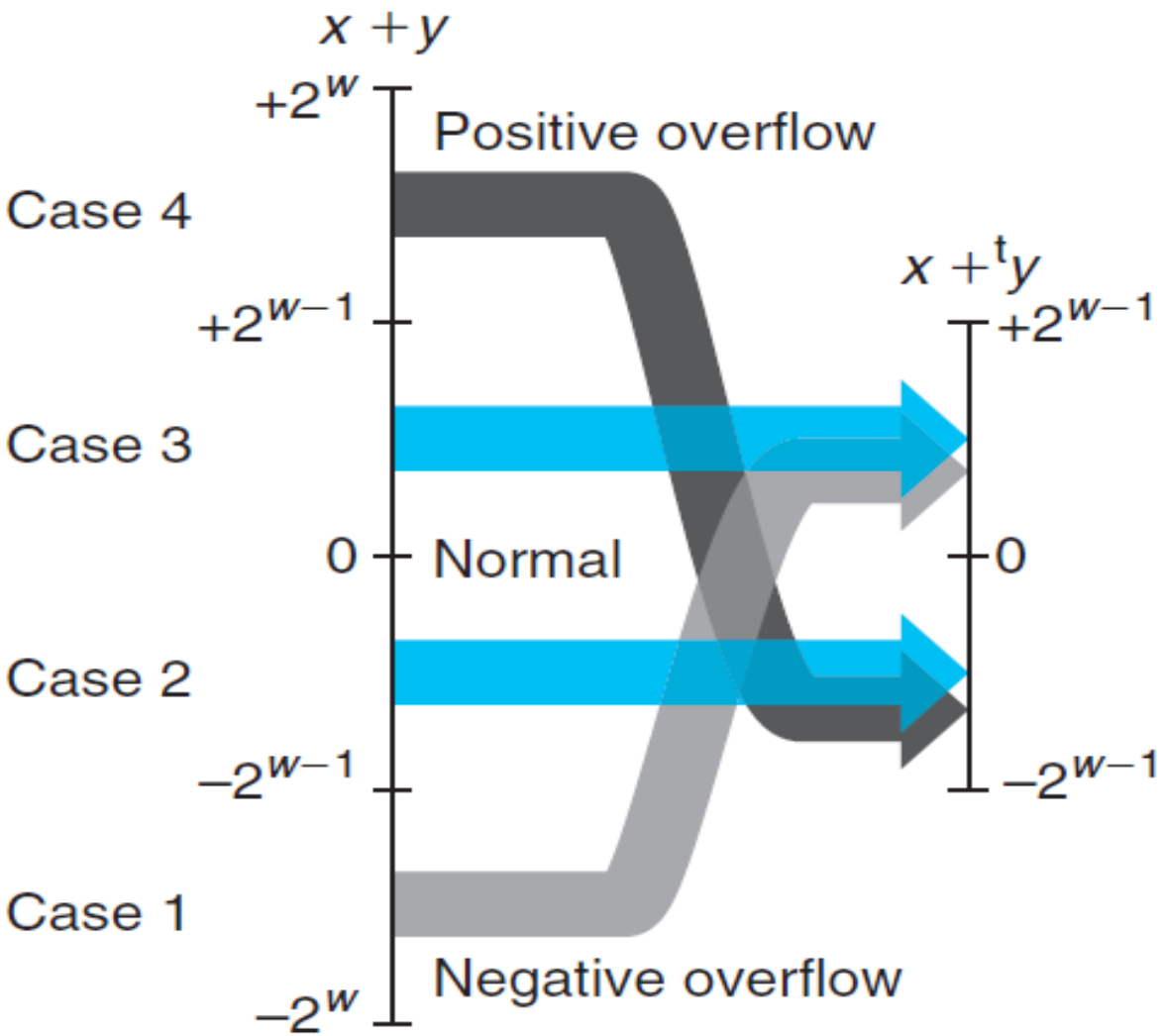
Signed Addition

- Functionality
  - 真正的结果需要w+1 bits
  - 但实际只有w bits, 会丢掉最高位
  - 将结果bits视为补码整型

$$Tadd(u,v) = \begin{cases} u+v-2^w, & TMax_w < u+v \ (PosOver) \\ u+v, & TMin_w \le u+v \le TMax_w \\ u+v+2^w, & u+v < TMin_w \ (NegOver) \end{cases}$$


Negative overflow
Normal
Positive overflow
8
6
4
2
0
−2
−4
−6
−8
−8
−6
−4
−2
0
2
4
6
−8
−6
−4
−2
0
2
4
6

Signed Addition



Detecting Tadd Overflow

- Task
  - Given $s = \text{TAdd}_w(u, v)$
  - Determine if $s = \text{Add}_w(u, v)$
- Claim
  - Overflow iff either:
    - $u, v < 0, s \geq 0$ (NegOver)
    - $u, v \geq 0, s < 0$ (PosOver)
  - ovf = (u<0 == v<0) && (u<0 != s<0);

Detecting Tadd Overflow

```
int tadd_ok_bugy(int x, int y)
{
        int sum = x + y ;
        return (sum-x == y) && (sum-y == x)
} // 问题在哪？
```

e.g., [-8, 7), x=4, y=5

//应该如何正确判断加法溢出？

Detecting Tadd Overflow

```
int tadd_ok (int x, int y)
{
        int sum = x + y ;
        return !((x>0&&y>0&&sum<0) ||
(x<0&&y<0&&sum>0))
}
```

Detecting Tsub Overflow

```
int tsub_ok(int x, int y)
{
        return tadd_ok(x, -y) ;
} // 问题在哪?
```

e.g., x>0, y=TMin; x<0, y=TMin; x=0, y=TMin

//写一个正确版本

Detecting Tsub Overflow

```
int tsub_ok(int x, int y)
{
        int diff = x-y;
        return !(x>=0&&y<0&&diff<0 ||
x<0&&y>0&&diff>0) // 0减正数 不会溢出
}
```

只有: 1)正数/0-负数，或2)负数-正数可能溢出

Mathematical Properties of TAdd

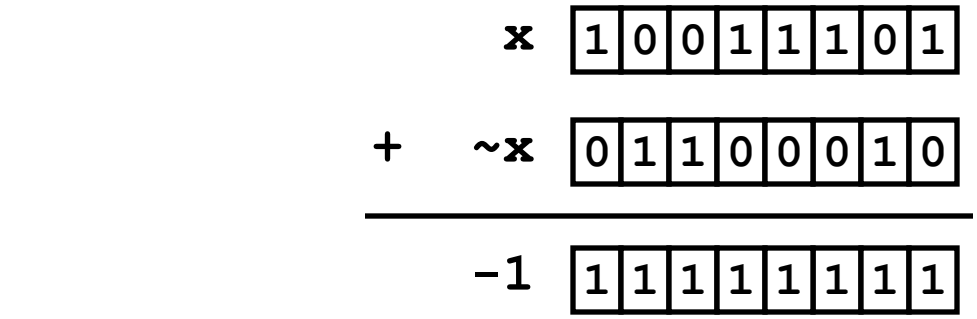
- Two's Complement Under TAdd Forms a Group
  - Closed, Commutative, Associative, 0 is additive identity
  - Every element has additive inverse
    - Let $$TComp_w(u) = \begin{cases} -u & u \neq TMin_w \\ TMin_w & u = TMin_w \end{cases}$$
    - $TAdd_w(u, TComp_w(u)) = 0$

Mathematical Properties of TAdd

- TAdd和Uadd：
  - $\text{TAdd}_w(u, v) = \text{U2T}(\text{UAdd}_w(\text{T2U}(u), \text{T2U}(v)))$
    - 因为二者有相同的位级表示
  - $\text{T2U}(\text{TAdd}_w(u, v)) = \text{UAdd}_w(\text{T2U}(u), \text{T2U}(v))$

Negating with Complement & Increment

- In C
  - ~x + 1 == -x
- Complement
  - Observation: ~x + x == 1111…111 == -1
- Increment
  - ~x + 1 == -x



Outline

- Integers
  - Representation: unsigned and signed
  - Conversion, casting
  - Expanding, truncating
  - Addition, negation, multiplication, shifting
  - **Summary**

Arithmetic: Basic Rules

- 加法：
  - Unsigned/signed: 先正常加，再截断，二者在bit级的操作完全一样
  - Unsigned: 相加后模2^w
    - 正常加法结果 + 可能减去2^w
  - Signed:相加后模2^w
    - 正常加法结果 + 可能加上或减去2^w

课堂练习1

- 库函数calloc有如下声明：
- Void *calloc (size_t nmemb, size_t size);
- 该函数为一个数组分配内存，该数组有nmemb个元素，每个元素为size字节，内存设置为0。如果nmemb或size为0，则返回NULL。
- 具体实现时，通过malloc分配内存，并用memset将内存设置为0.
- 你的代码应该没有任何由算术溢出引发的漏洞
  - Void *malloc (size_t size);
  - Void *memset(void *s, int c, size_t n);

- void *calloc(size_t nmemb, size_t size) {
  - size_t asize = nmemb * size; /* Check for overflow */
  - if (nmemb == 0 || size == 0 || asize / nmemb != size) /* Error */
    - return NULL;
  - void *result = malloc(asize);
  - if (result != NULL) {
    - memset(result, 0, asize);
    - return result;
  - }
  - return NULL;
- }

课堂练习3

- Int x = random();
- Int y = random();
- Unsigned ux = (unsigned)x;
- Unsigned uy = (unsigned)y;
- Int为32位，对于下列每个C表达式，你要指出其是否总为1。如果是，请指出其数学原理；否则，举反例
  1. (x<y) == (-x>-y)
  2. ((x+y)<<4) + y -x == 17*y + 15*x
  3. ~x+~y+1 == ~(x+y)
  4. (ux-uy) == -(unsigned)(y-x)
  5. ((x>>2)<<2) <= x

课堂练习3

1. (x<y) == (-x>-y)
2. ((x+y)<<4) + y -x == 17*y + 15*x
3. ~x+~y+1 == ~(x+y)
4. (ux-uy) == -(unsigned)(y-x)
5. ((x>>2)<<2) <= x

1. 否，当x=Tmin时不成立，-x=Tim，还是小于-y
2. 是，补码的循环特性，即使一边溢出，或两边都溢出，最终左右还是会相等
3. 是，~x+1+~y+1=-x-y=-(x+y)=~(x+y)+1
4. 是，有符号和无符号的加减法在binary层级是完全一样的，这个表达式相当于x==-(-x)是否成立？X=Tmin也是成立的。
5. 是，x为正数时，可能损失最后两位，变小；x为负数时，binary变小，值变小（绝对值变大）【补码映射是线性关系，增大变小的关系，正负数是一样的】

课堂练习4

- 你刚刚开始在一家公司工作，他们要用到一个数据结构，这个结构是将4个有符号字节封装为一个32位的unsigned。一个32位字中的字节从0（最低有效字节）编号到3（最高有效字节）。分配给你的任务是：编写一个函数，提取其中的按照制定的bytenum从中提取需要的有符号字节，将其符号扩展为一个32位int：
- ```
  typedef unsigned packed_t;
  /* Extract byte from word. Return as signed integer */
  int xbyte (packed_t word, int bytenum);
  ```
- 你的前任（因为水平不够高被解雇了）编写了下面的代码：
- ```
  /* Failed attempt at xbyte */
  int xbyte (packed_t word, int bytenum) {
          return (word >> (bytenum << 3)) & 0xFF;
  }
  ```
- 这段代码错在哪里？
- 给出函数的正确实现，除了赋值语句外，只能使用左右移位和一个减法。