



信息的表示和处理 (2)

王晶

jwang@ruc.edu.cn, 信息楼124

2025年10月



提纲

- 整数
 - 表示：有符号数和无符号数
 - 转换
 - 扩展和截断
 - 计算
 - 总结



无符号数

- 二进制 (物理存储形式)
 - 位向量 $[x_{w-1}, x_{w-2}, x_{w-3}, \dots, x_0]$
- 二进制转换成无符号数Unsigned(逻辑表达)

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

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**。



无符号数

- 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 $0 \sim 255$ 。
- 计算机的内存地址就是无符号数的例子。

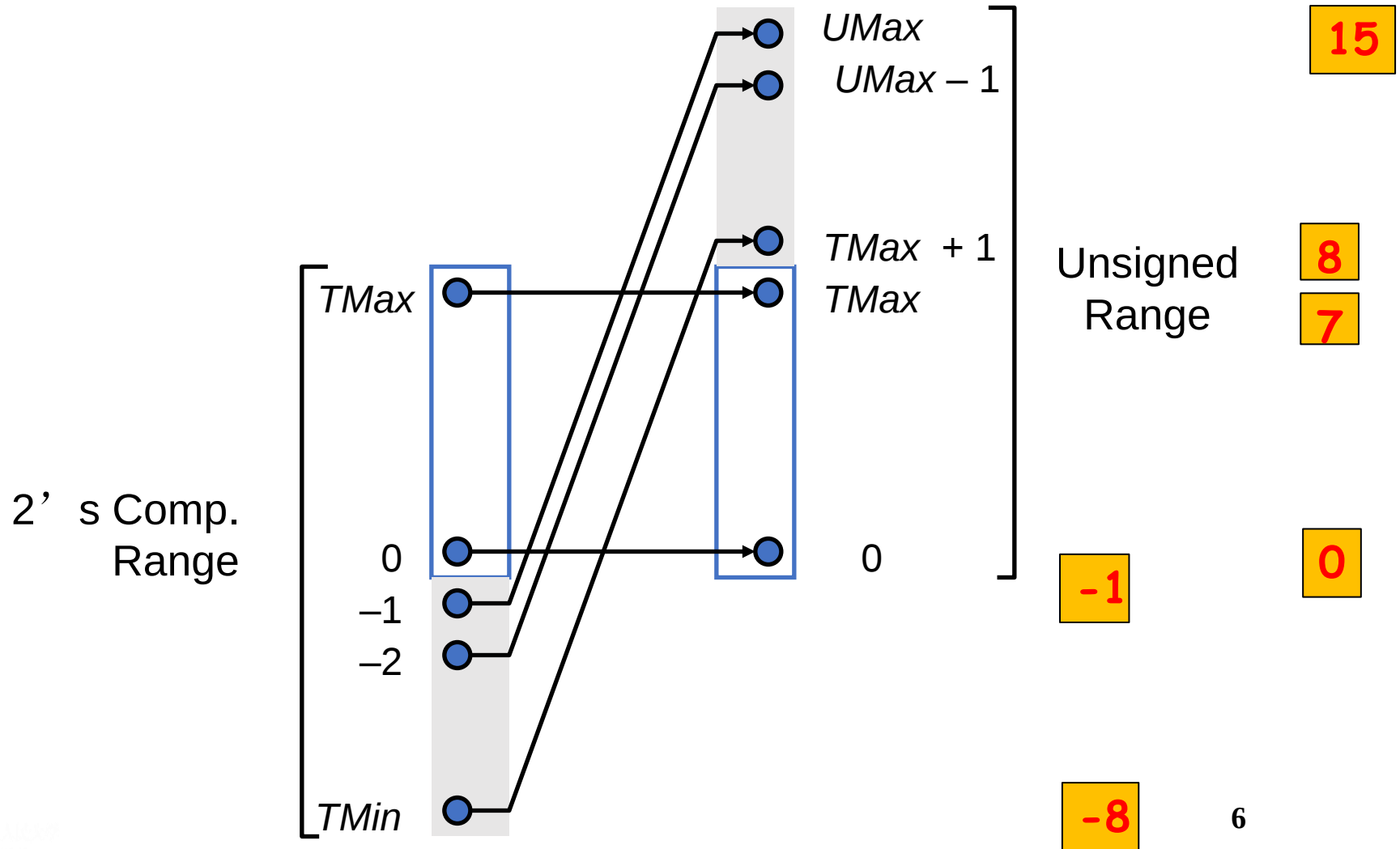


有符号数Signed Representation

- 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





Two-complement补码

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

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

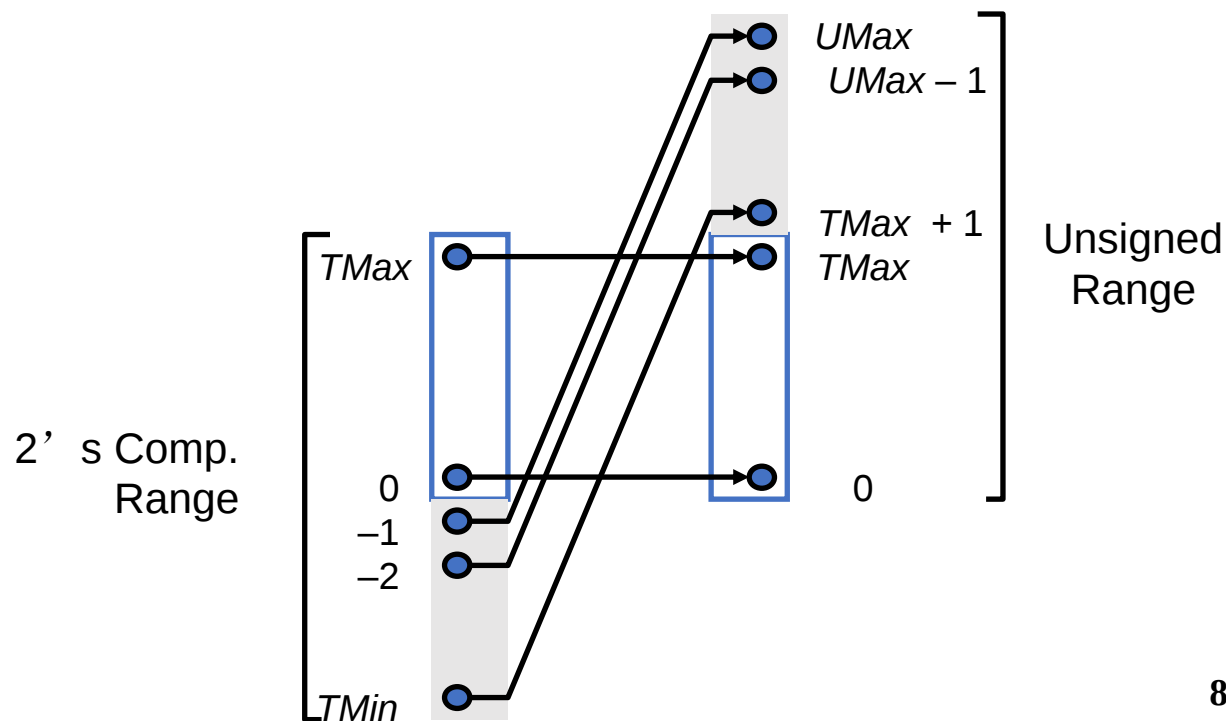
$$10 = \begin{array}{rcccll} & -16 & 8 & 4 & 2 & 1 \\ & 0 & 1 & 0 & 1 & 0 \end{array} \quad 8+2 = 10$$

$$-10 = \begin{array}{rcccll} & -16 & 8 & 4 & 2 & 1 \\ & 1 & 0 & 1 & 1 & 0 \end{array} \quad -16+4+2 = -10$$



Two's Complement

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





补码 - 模运算 (modular运算)

重要概念： 在一个模运算系统中，一个数与它除以“模”后的余数等价。

时钟是一种模12系统 现实世界中的模运算系统

假定钟表时针指向10点，要将它拨向6点， 则有两种拨法：

① 倒拨4格： $10 - 4 = 6$

② 顺拨8格： $10 + 8 = 18 \equiv 6 \pmod{12}$

模12系统中： $10 - 4 \equiv 10 + 8 \pmod{12}$

$-4 \equiv 8 \pmod{12}$

则，称8是-4对模12的补码（即：-4的模12补码等于8）。

同样有 $-3 \equiv 9 \pmod{12}$

$-5 \equiv 7 \pmod{12}$ 等

结论1： 一个负数的补码等于模减该负数的绝对值。

结论2： 对于某一确定的模，某数减去小于模的另一数，总可以用该数加上另一数负数的补码来代替。

补码 (modular运算)：+ 和- 的统一



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： $9 - 5 = 9 + (12 - 5) = 9 + 7 = 4 \pmod{12}$

$$65 - 25 = 65 + (-25) = 65 + (100 - 25) = 65 + 75 = 40 \pmod{100}$$



求补码的快捷方式

设 $x = -1010$ 时

$$\begin{aligned} \text{则 } [x]_{\text{补}} &= 2^{4+1} - 1010 &= 11111 + 1 - 1010 \\ &= 100000 &= 11111 + 1 \\ &\quad - 1010 &\quad - 1010 \\ \hline &= 1,0110 &\quad 10101 + 1 \\ & &= 1,0110 \end{aligned}$$

当真值为 负 时

每位取反，末位加 1 求得



From a Number to Two's Complement

- 真值-5
 - $y=5$
 - 0101 (binary for 5)
 - 1010 (after complement)
 - 1011 (add 1)
 - 补码_x为1011
- 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 Ranges

- 无符号数 Unsigned Values

- $UMin = 0$

000...0

- $UMax = 2^w - 1$

111...1

- 补码 Two's Complement Values

- $TMin = -2^{w-1}$

100...0

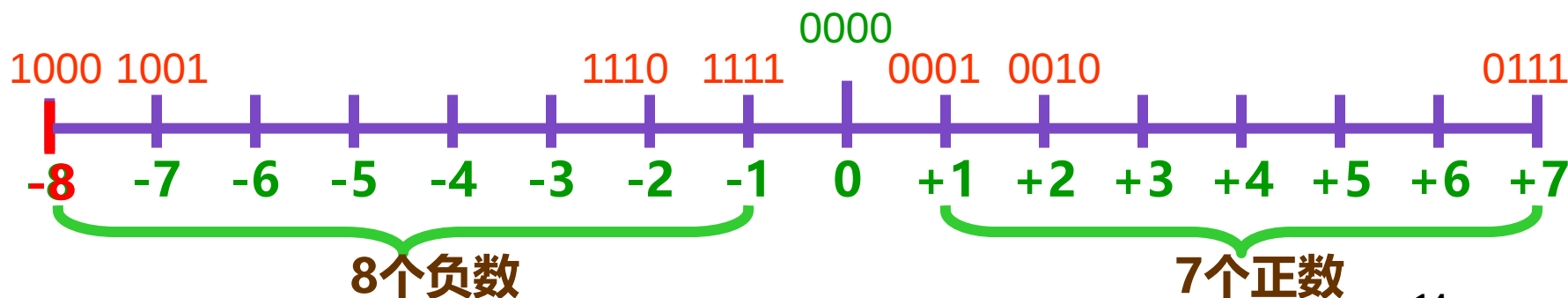
- $TMax = 2^{w-1} - 1$

011...1

E.g., $w=4$

0000 ~ 0111: 0~7

1000 ~ 1111: -8~-1





Numeric Ranges

- Unsigned Values

- $UMin = 0$
000...0
- $UMax = 2^w - 1$
111...1

- Two's Complement Values

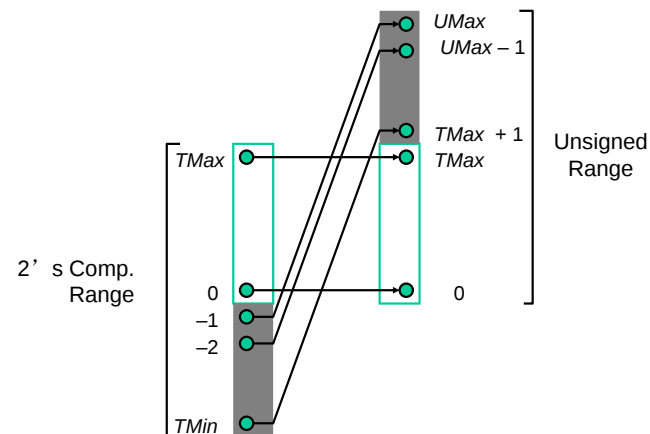
- $TMin = -2^{w-1}$
100...0
- $TMax = 2^{w-1} - 1$
011...1

- Other Values

- Minus 1
111...1

Values for $W = 16$

	Decimal	Hex	Binary
UMax	65535	FF FF	11111111 11111111
TMax	32767	7F FF	01111111 11111111
TMin	-32768	80 00	10000000 00000000
-1	-1	FF FF	11111111 11111111
0	0	00 00	00000000 00000000





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



有符号数的多种表示方式

➤ 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

➤ 对于正数，原码 = 补码 = 反码

➤ 对于负数，符号位为 1，其数值部分

原码除符号位外每位取反末位加 1 → 补码

原码除符号位外每位取反 → 反码

写出下列数字的补码

(1)+0.1101B的补码是 [填空1] B

(2)-0.1101B的补码是[填空2] B

(3)-178的补码是[填空3] H (short类型)

(4)-39的补码是[填空4] H (short类型)

(5)-26的补码是[填空5] H (short类型)

下面给出了一组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)。

(1)FF3C (short类型) 的真值是 [填空1]

(2)FF86 (short类型) 的真值是 [填空2]

(3)char x = 10110111b, x的真值是 [填空3]



提纲

- 整数
 - 表示：有符号数和无符号数
 - 转换
 - 扩展和截断
 - 计算
 - 总结



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



Casting

- 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();`



Mapping Signed $\leftrightarrow$ Unsigned

Bits	Signed		Unsigned
0000	0	T2U U2T	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	-7		9
1010	-6		10
1011	-5		11
1100	-4		12
1101	-3		13
1110	-2		14
1111	-1		15



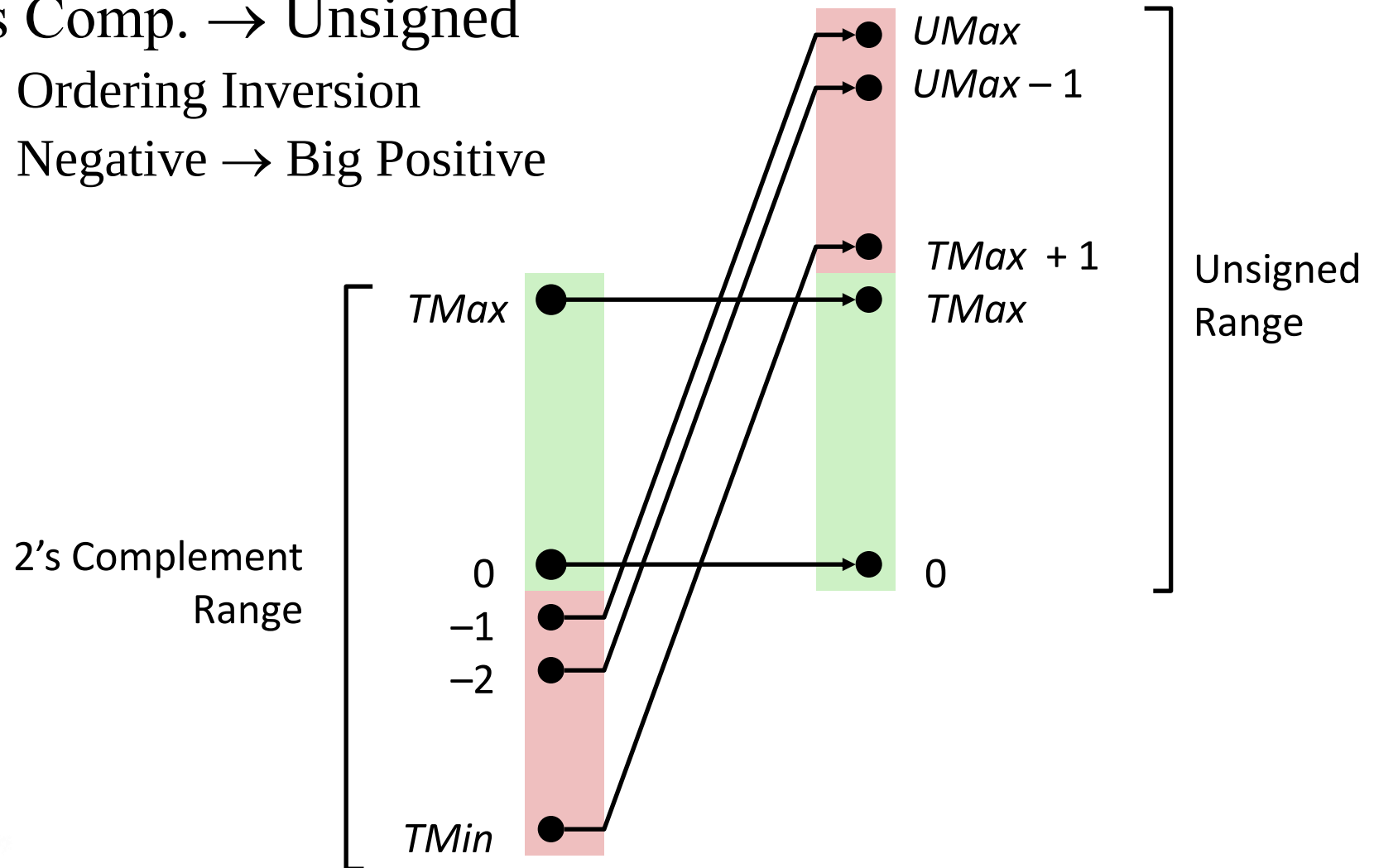
Mapping Signed $\leftrightarrow$ Unsigned

Bits	Signed		Unsigned
0000	0	$\longleftrightarrow$ =	0
0001	1		1
0010	2		2
0011	3		3
0100	4		4
0101	5		5
0110	6		6
0111	7		7
1000	-8	$\longleftrightarrow$ +/- 16	8
1001	-7		9
1010	-6		10
1011	-5		11
1100	-4		12
1101	-3		13
1110	-2		14
1111	-1		15



Conversion Visualized

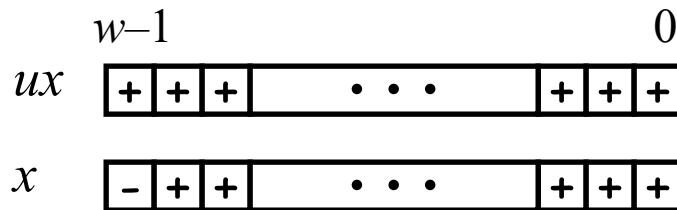
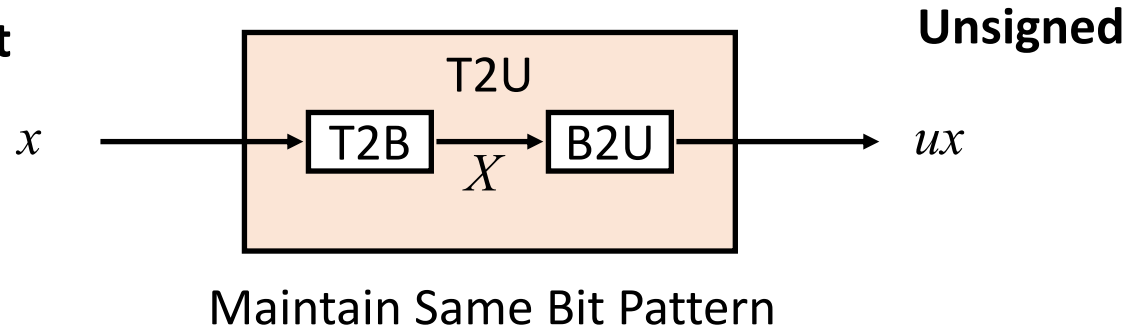
- 2's Comp. $\rightarrow$ Unsigned
 - Ordering Inversion
 - Negative $\rightarrow$ Big Positive





Relation between Signed & Unsigned

Two's Complement

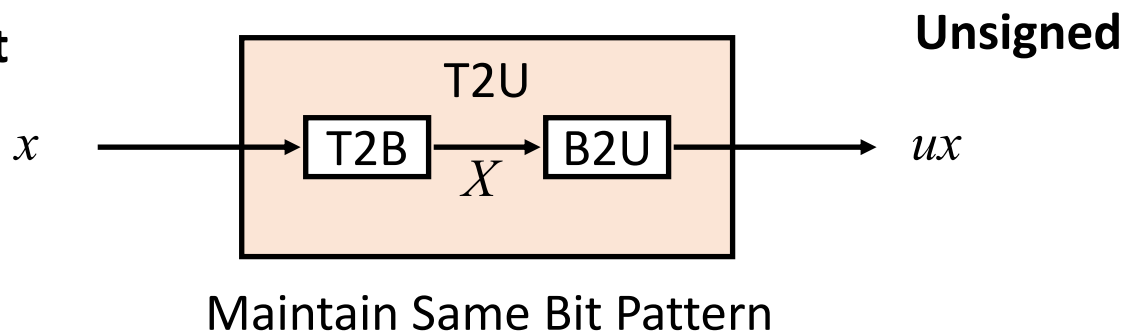


Large negative weight
becomes
Large positive weight

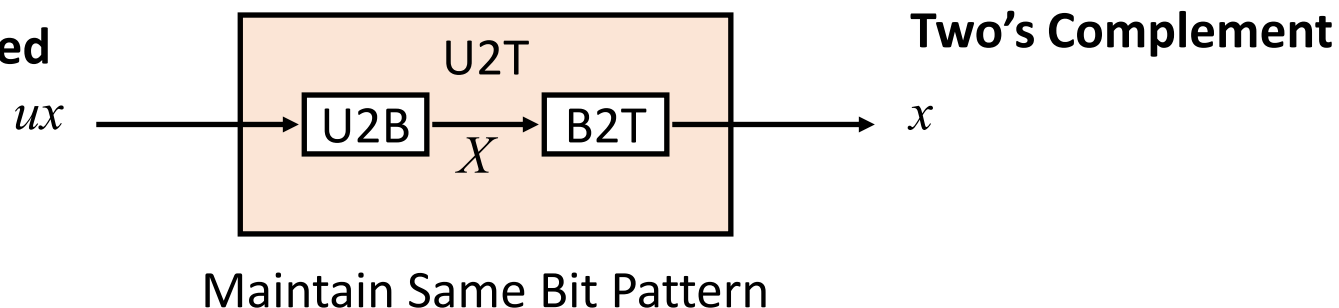


Mapping Between Signed & Unsigned

Two's Complement



Unsigned



- Mappings between unsigned and two's complement numbers:
keep bit representations and reinterpret



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
 - 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



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

```
unsigned int i
for (i=n-1; i>= 0; i--)
```

```
for (i=n-1; i-sizeof (char) >= 0; i--)
```

```
int array[] = {1,2,3};
#define TOTAL sizeof(array) /* unsigned int */
void main() {
    int d = -1;
    if (d <= TOTAL)
        printf("small\n");
    else printf("large\n");
}
```



Casting Surprises

- Expression Evaluation

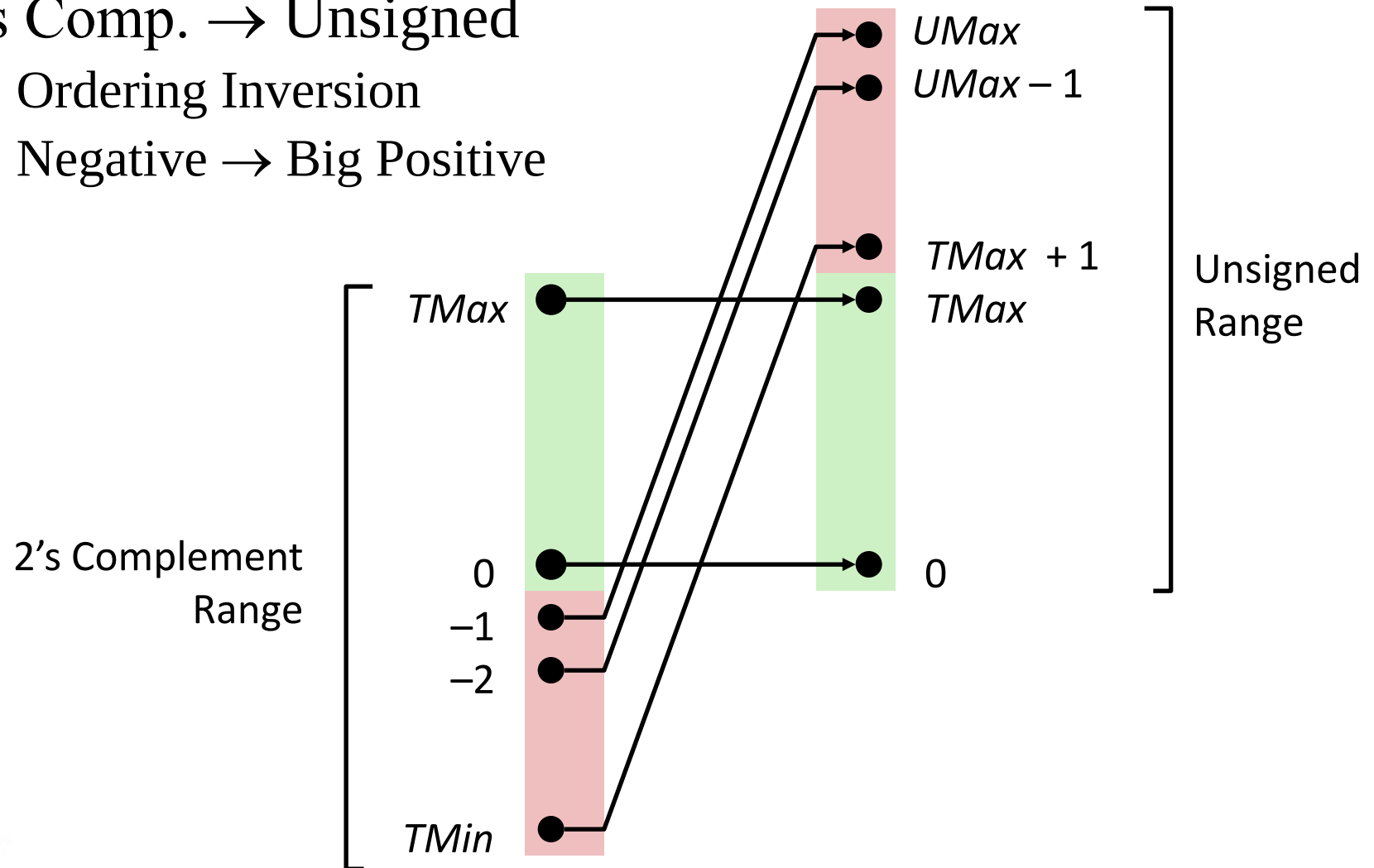
- If there is a mix of unsigned and signed in single expression, signed values implicitly cast to unsigned
- Including comparison operations $<$, $>$, $==$, $<=$, $>=$
- Examples for $W = 32$: $TMIN = -2,147,483,648$, $TMAX = 2,147,483,647$

Constant1	Constant2	Relation	Evaluation
• 0	0U		
• -1	0		
• -1	0U		
• 2147483647	-2147483647-1		
• 2147483647U	-2147483647-1		
• -1	-2		
• (unsigned)-1	-2		
• 2147483647	2147483648U		
• 2147483647	(int) 2147483648U		



Conversion Visualized

- 2's Comp. $\rightarrow$ Unsigned
 - Ordering Inversion
 - Negative $\rightarrow$ Big Positive





Casting Surprises

- Expression Evaluation
 - If there is a mix of unsigned and signed in single expression, *signed values implicitly cast to unsigned*
 - Including comparison operations $<$, $>$, $==$, $<=$, $>=$
 - Examples for $W = 32$: **TMIN = -2,147,483,648 , TMAX = 2,147,483,647**

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



提纲

- 整数

- 表示：有符号数和无符号数

- 转换

- 扩展和截断

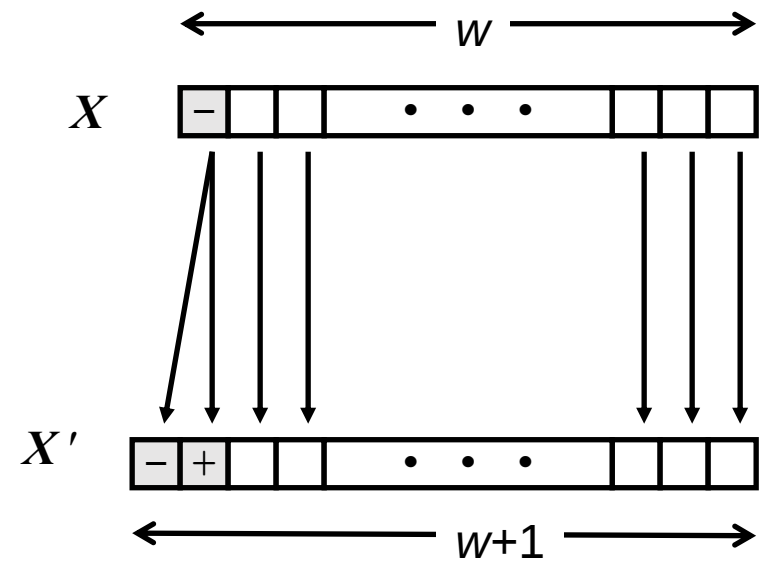
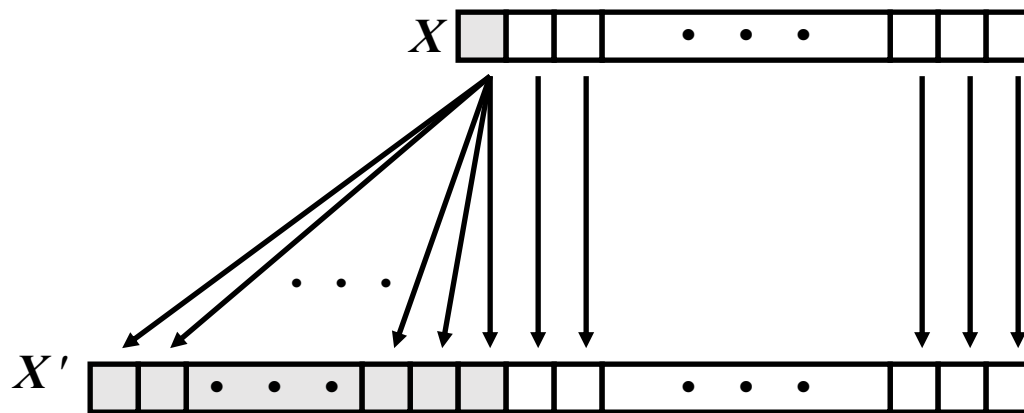
- 计算

- 总结



Expanding the Bit Representation

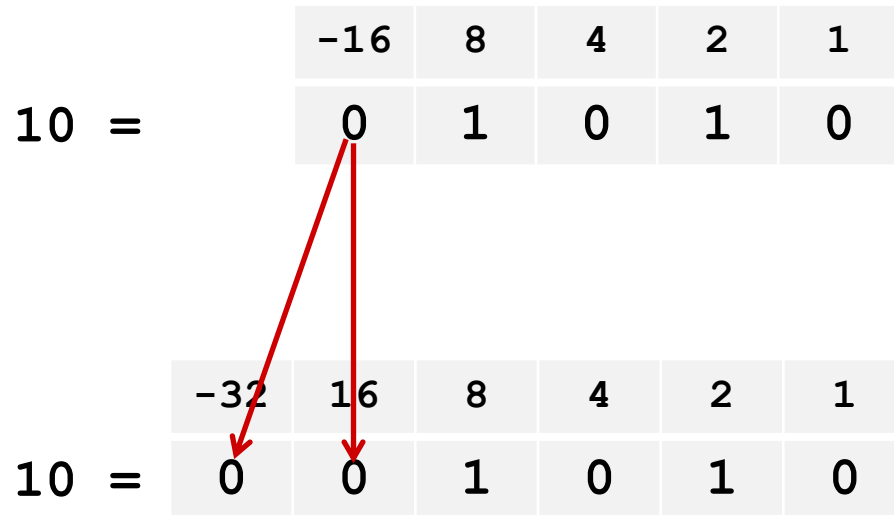
- Zero extension
 - 在前面添加0
- Sign extension
 - $[x_{w-1}, x_{w-2}, x_{w-3}, \dots, x_0]$



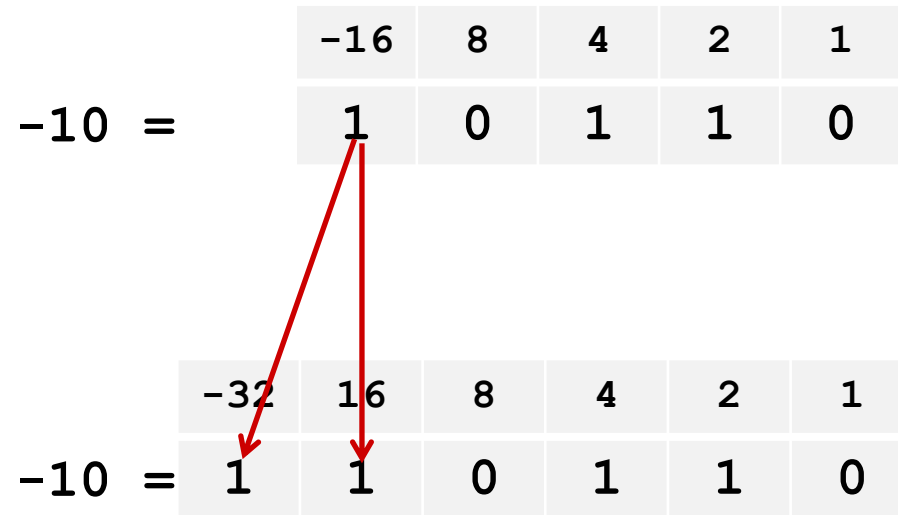


Sign Extension: Simple Example

Positive number



Negative number





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



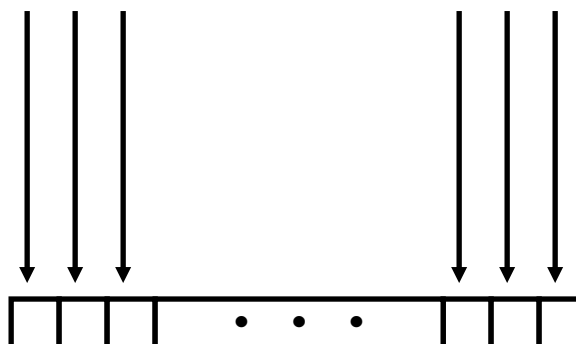
Truncating Numbers

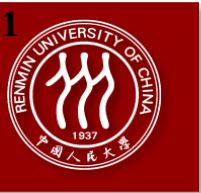
	Decimal	Hex	Binary
x	53191	00 00 CF C7	00000000 00000000 11001111 11000111
sx	-12345	CF C7	11001111 11000111
y	-12345	FF FF CF C7	11111111 11111111 11001111 11000111
sy	-12345	CF C7	11001111 11000111

X'



X





Truncating Numbers

- Unsigned Truncating

$$B2U_w([x_w, x_{w-1}, \dots, x_0]) \bmod 2^k = B2U_k([x_k, x_{k-1}, \dots, x_0])$$

- Signed Truncating

- 先当作无符号数去截断，然后转换为有符号数

$$B2T_k([x_k, x_{k-1}, \dots, x_0]) = U2T_k(B2U_w([x_w, x_{w-1}, \dots, x_0]) \bmod 2^k)$$



Truncation: Simple Example

No sign change

	-16	8	4	2	1
2 =	0	0	0	1	0

	-8	4	2	1
2 =	0	0	1	0

$$2 \bmod 16 = 2$$

	-16	8	4	2	1
-6 =	1	1	0	1	0

	-8	4	2	1
-6 =	1	0	1	0

$$-6 \bmod 16 = 26U \bmod 16 = 10U = -6$$

Sign change

	-16	8	4	2	1
10 =	0	1	0	1	0

	-8	4	2	1
-6 =	1	0	1	0

$$10 \bmod 16 = 10U \bmod 16 = 10U = -6$$

	-16	8	4	2	1
-10 =	1	0	1	1	0

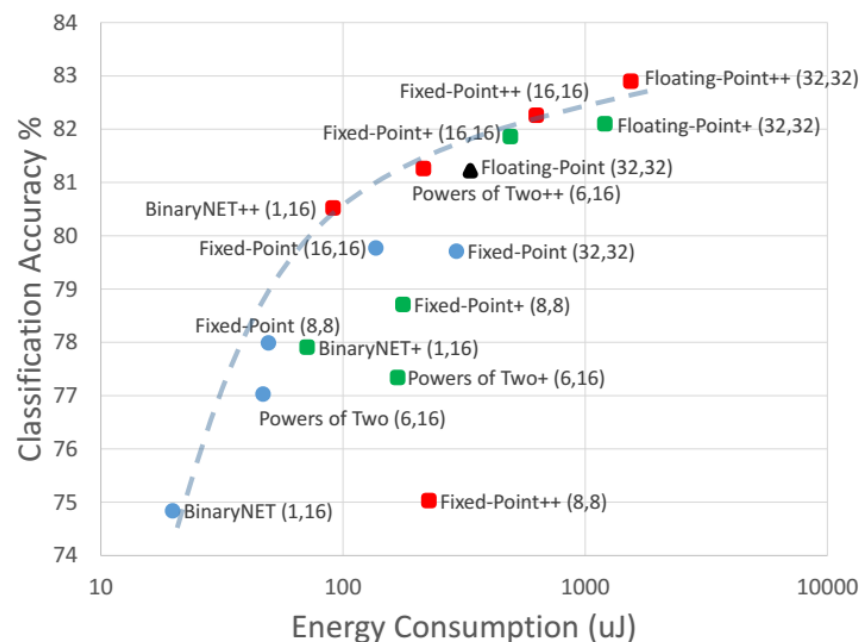
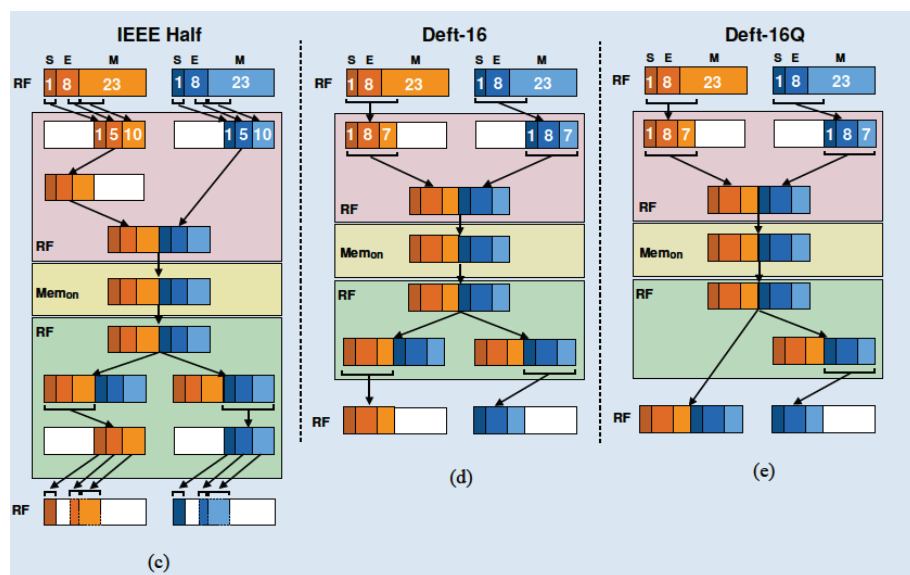
	-8	4	2	1
6 =	0	1	1	0

$$-10 \bmod 16 = 22U \bmod 16 = 6U = 6$$



截断在人工智能中的应用

- 精度缩放，压缩数据减少计算和存储开销
- 精度损失很小，几乎可以忽略不计
- 典型的深度学习模型用**8位**数据就可以达到**95%**的精度





Advice on Signed vs. Unsigned

```
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 */*

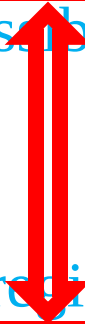
/ WARNING: This function is buggy */*

```
int strlonger(char *s, char *t) {  
    return strlen(s) - strlen(t) > 0;  
}
```



FreeBSD's implementation of getpeername (...)

```
/* Declaration of library function memcpy */
void *memcpy(void *dest, void *src, size_t n);
/* Kernel memory region holding user-accessible data */
#define KSIZE 1024u
char kbuf[KSIZE];
/* Copy at most maxlen bytes from kernel region to user buffer */
int copy_from_kernel (void *user_dest, int maxlen) {
/* Byte count len is minimum of buffer size and maxlen */
int len = KSIZE < maxlen ? KSIZE : maxlen;
memcpy(user_dest, kbuf, len); // 如果maxlen是个负值
return len; }
```





提纲

- 整数

- 表示：有符号数和无符号数

- 转换

- 扩展和截断

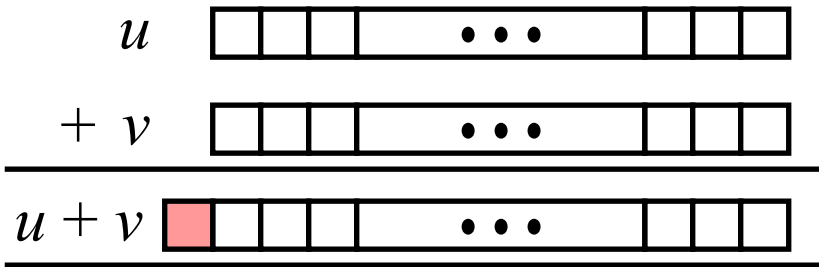
- 计算

- 总结



无符号加法

Operands: w bits



True Sum: $w+1$ bits

Discard Carry: w bits $\text{UAdd}_w(u, v)$

- Standard Addition Function
 - Ignores carry output
- Implements Modular Arithmetic

$$s = \text{UAdd}_w(u, v) = u + v \bmod 2^w$$

unsigned char

1110 1001

+

1101 0101

=====

E9

+

D5

=====

233

+

213

=====

Hex	Decimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

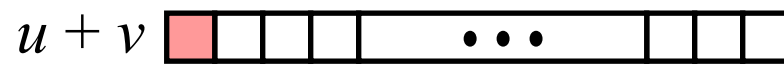


无符号加法

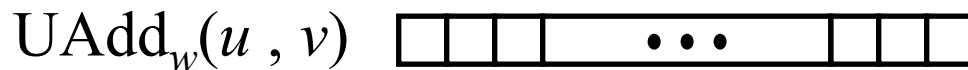
Operands: w bits



True Sum: $w+1$ bits



Discard Carry: w bits



- Standard Addition Function

- Ignores carry output

- Implements Modular Arithmetic

$$s = \text{UAdd}_w(u, v) = u + v \bmod 2^w$$

unsigned char

$$\begin{array}{r} 1110\ 1001 \\ +\ 1101\ 0101 \\ \hline 1\ 1011\ 1110 \\ \hline 1011\ 1110 \end{array}$$

$$\begin{array}{r} \text{E9} \\ +\ \text{D5} \\ \hline \text{1BE} \\ \hline \text{BE} \end{array}$$

$$\begin{array}{r} 233 \\ +\ 213 \\ \hline 446 \\ \hline 190 \end{array}$$

Hex	Decimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

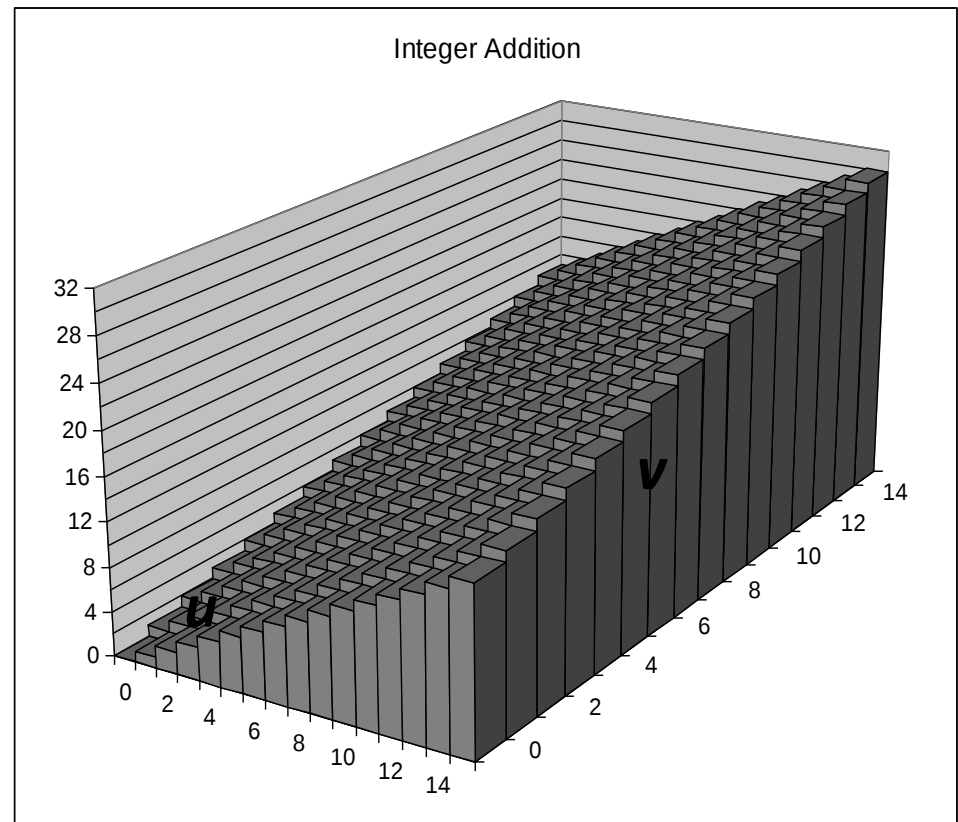


加法结果可视化效果

- Integer Addition

- 4-bit integers u, v
- Compute true sum $\text{Add}_4(u, v)$
- Values increase linearly with u and v
- Forms planar surface

$$\text{Add}_4(u, v)$$

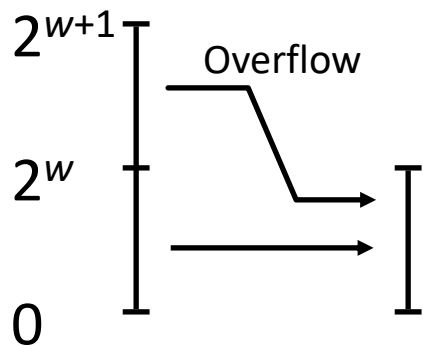




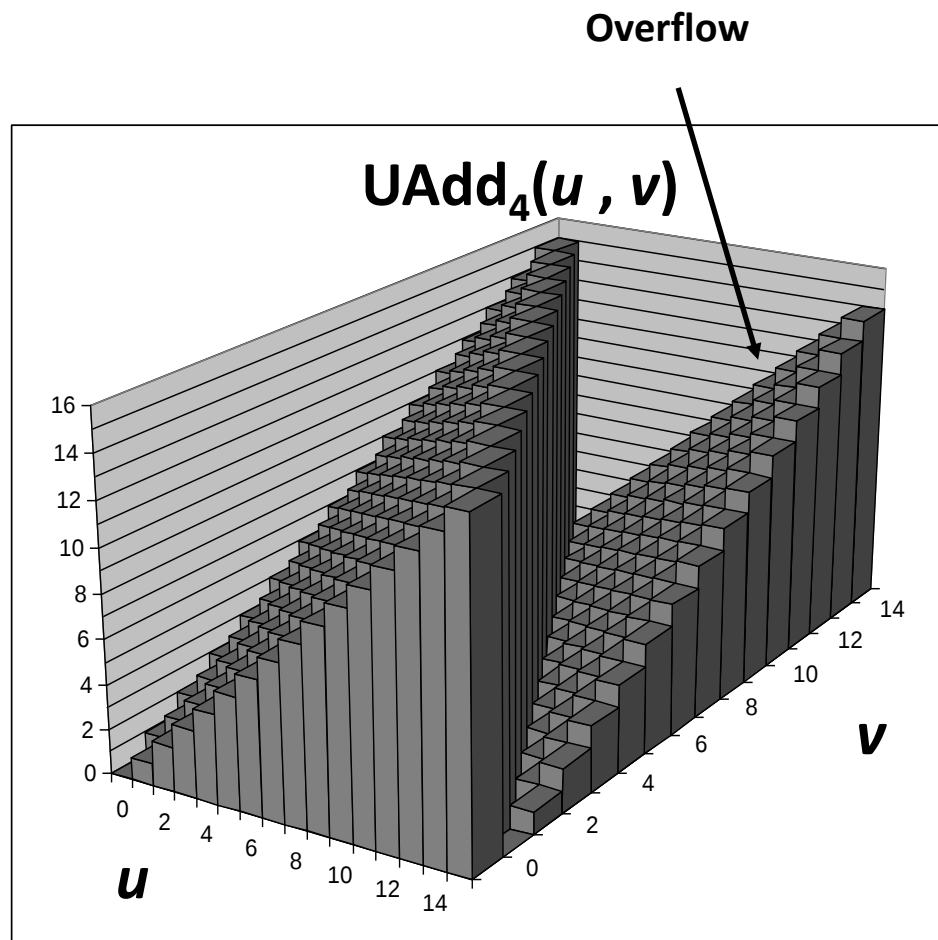
加法结果可视化效果

- Wraps Around
 - If true sum $\geq 2^w$
 - At most once

True Sum



Modular Sum





无符号加法

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 if $(X+Y) < X$



无符号加法形成阿贝尔群

- 加法结果是封闭的（还在同样的数据范围内）
 - $0 \leq \text{UAdd}_w(u, v) \leq 2^w - 1$
- 交换律
 - $\text{UAdd}_w(t, \text{UAdd}_w(u, v)) = \text{UAdd}_w(\text{UAdd}_w(t, u), v)$
- 有一个单位元0
 - $\text{UAdd}_w(u, 0) = u$
- 每个元素都有一个加法逆元
 - Let $\text{UComp}_w(u) = 2^w - u$
 - $\text{UAdd}_w(u, \text{UComp}_w(u)) = 0$



$\mathcal{U}$

$$+ \quad v$$
$$u + v \quad \boxed{\text{red}} \boxed{} \boxed{} \boxed{} \boxed{ \cdots } \boxed{} \boxed{} \boxed{}$$
$$\text{TAdd}_w(u, v) \quad \boxed{} \boxed{} \boxed{} \boxed{} \cdots \boxed{} \boxed{} \boxed{}$$

- Will give $s == t$

1110 1001	E9	-23
+ 1101 0101	+ D5	+ -43
1 1011 1110	1BE	-66
1011 1110	BE	-66



补码加法的溢出

位宽是4，其中1位符号位，3位数值位

$$\begin{array}{r} 5 \\ + 4 \\ \hline 9 \end{array}$$

$$\begin{array}{r} 5 \\ + (-4) \\ \hline 1 \end{array}$$

$$\begin{array}{r} -5 \\ + 4 \\ \hline -1 \end{array}$$

$$\begin{array}{r} -5 \\ + (-4) \\ \hline -9 \end{array}$$

$$\begin{array}{r} 0101 \\ + 0100 \\ \hline 1001 \\ -7 \end{array}$$

$$\begin{array}{r} 0101 \\ + 1100 \\ \hline 0001 \\ 1 \end{array}$$

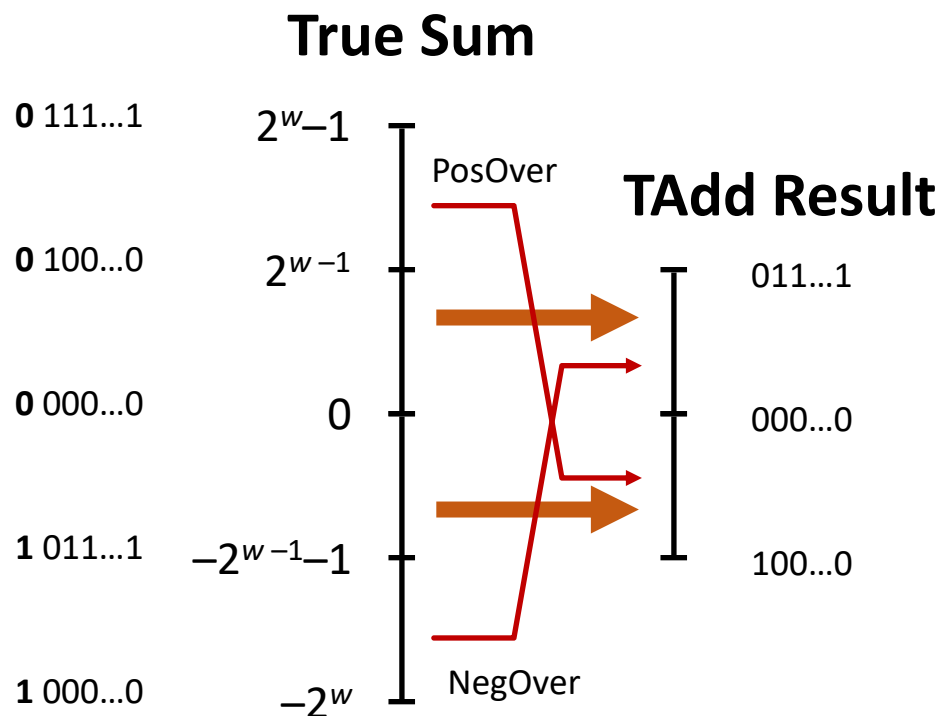
$$\begin{array}{r} 1011 \\ + 0100 \\ \hline 1111 \\ -1 \end{array}$$

$$\begin{array}{r} 1011 \\ + 1100 \\ \hline 0111 \\ 7 \end{array}$$



补码加法溢出

- Functionality
 - True sum requires $w+1$ bits
 - Drop off MSB
 - Treat remaining bits as 2's comp. integer

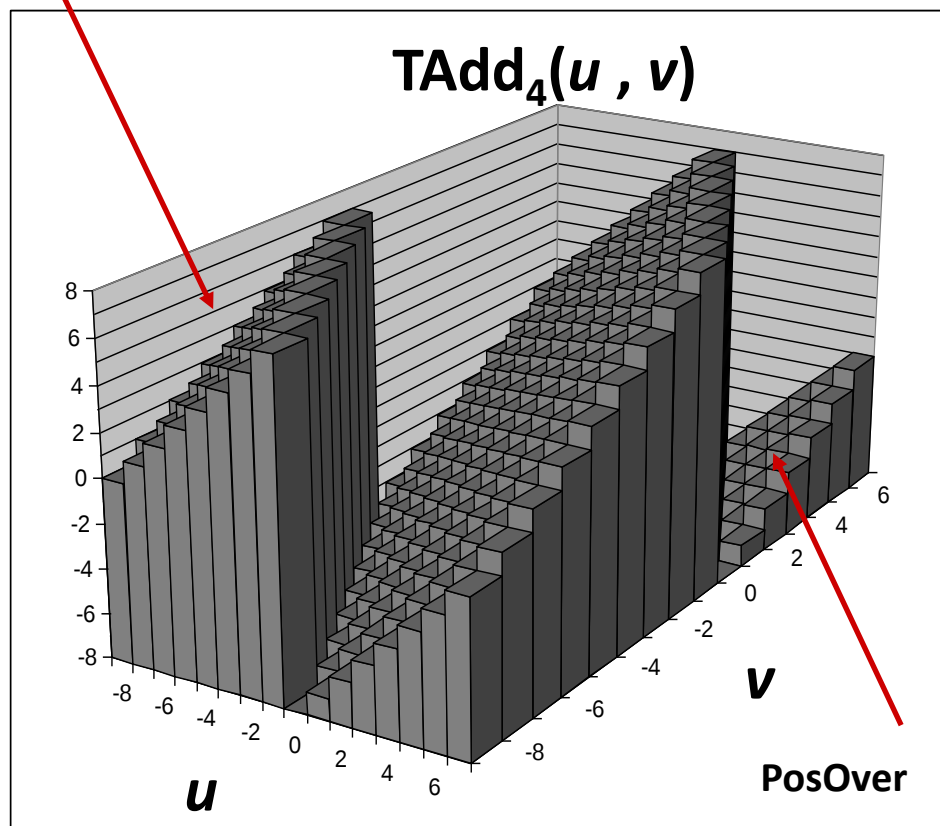




补码加法可视化效果

- Values
 - 4-bit two's comp.
 - Range from -8 to +7
- Wraps Around
 - If $\text{sum} \geq 2^{w-1}$
 - Becomes negative
 - At most once
 - If $\text{sum} < -2^{w-1}$
 - Becomes positive
 - At most once

NegOver

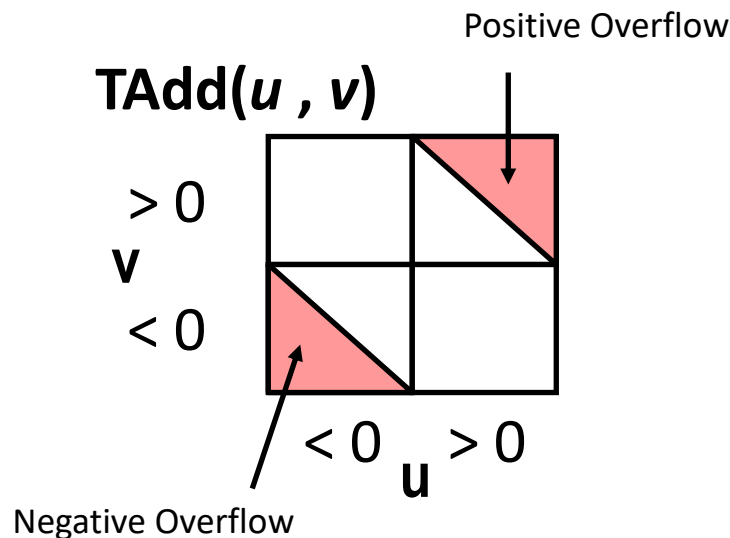




补码加法

- Functionality

- True sum requires $w+1$ bits
- Drop off MSB
- Treat remaining bits as 2's comp. integer

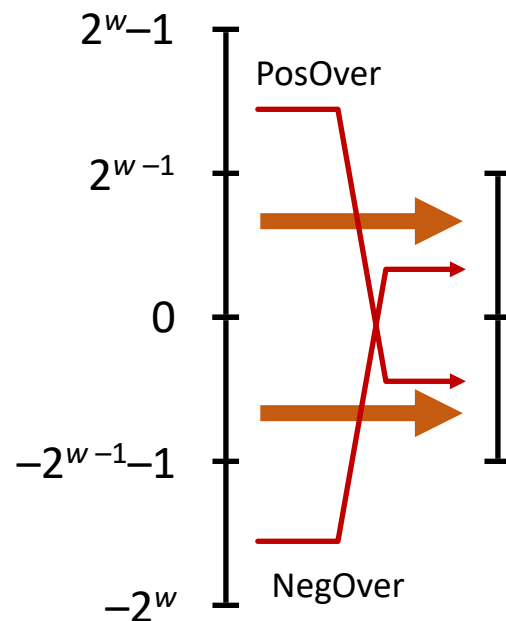


$$TAdd_w(u, v) = \begin{cases} u + v + 2^w & u + v < TMin_w \text{ (NegOver)} \\ u + v & TMin_w \leq u + v \leq TMax_w \\ u + v - 2^w & TMax_w < u + v \text{ (PosOver)} \end{cases}$$



检测补码加法溢出

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





检测补码加法溢出

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

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

$sum=9=-7$. $sum-x = -7-4=-11=5=y$ (-11 还是会被拉回到5)

tadd是阿贝尔群，结果封闭（越界则加上或减少 2^w ），所以 $sum-x = x+y-x$ 永远等于 y （因为 y 就在合法区间，不会加减 2^w ）。此函数无法判定溢出。

//应该如何正确判断加法溢出？



检测补码加法溢出

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



检测补码减法溢出

```
int tsub_ok(int x, int y)
```

```
{
```

```
    return tadd_ok(x, -y);
```

```
} // 问题在哪?
```

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

- 如果 x 是正数，判定是异号相加，肯定不溢出，但实际是正数 $-Tmin$ ，肯定溢出。
- 如果 x 是负数，会判定溢出，实际不溢出。
- 如果 $x = 0$ ，会判定不溢出，实际溢出。



检测补码减法溢出

```
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-正数 不会溢出
}
```



补码加法的数学特性

- Two's Complement Under TAdd Forms a Group
 - Closed, Commutative, Associative, 0 is additive identity
 - Every element has additive inverse
 - Let
 - $TAdd_w(u, TComp_w(u)) = 0$

$$TComp_w(u) = \begin{cases} -u & u \neq TMin_w \\ TMin_w & u = TMin_w \end{cases}$$



补码加法的数学特性

- TAdd和Uadd:
 - $TAdd_w(u, v) = U2T(UAdd_w(T2U(u), T2U(v)))$
 - 因为二者有相同的位级表示
 - $T2U(TAdd_w(u, v)) = UAdd_w(T2U(u), T2U(v))$

补码：连同符号位一起运算



-x的补码

- In C

- $\sim x + 1 == -x$

- Complement

- Observation: $\sim x + x == 1111...111 == -1$

- Increment

- $\sim x + 1$
 - $== \sim x + x + (-x + 1)$
 - $== -1 + (-x + 1)$
 - $== -x$

$$\begin{array}{r} x \quad 1\ 0\ 0\ 1\ 1\ 1\ 0\ 1 \\ + \quad \sim x \quad 0\ 1\ 1\ 0\ 0\ 0\ 1\ 0 \\ \hline -1 \quad 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1 \end{array}$$

x为负数时：

x的补码是原码除符号位外，每位取反，末位加1

-x的补码是原码连同符号位，每位取反，末位加1



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2

- `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. $((x+y) \ll 4) + y - x == 17*y + 15*x$
 3. $\sim x + \sim y + 1 == -(x+y)$
 4. $(ux - uy) == -(\text{unsigned})(y - x)$
 5. $((x \gg 2) \ll 2) \leq x$



课堂练习2

1. $(x < y) == (-x > -y)$

否，当 $x = T_{min}$ 时不成立， $-x = T_{min}$ ，还是小于 $-y$

2. $((x+y) << 4) + y - x == 17*y + 15*x$

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. $\sim x + \sim y + 1 == \sim(x+y)$

是， $\sim x + 1 + \sim y + 1 = -x - y = -(x+y) = \sim(x+y) + 1$

4. $(ux - uy) == -(\text{unsigned})(y - x)$

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？ $X = T_{min}$ 也是成立的。

5. $((x >> 2) << 2) <= x$

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3

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，提取其中的按照指定的**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？

```
( (int)word << ((3-bytenum)<<3)) >> 24
```

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。