

# Part1: 系统中的数据表示和处理 (比特/位 **Bits**、字节 **Bytes**, 和整数 **Integers**)

人工智能学院:计算机原理与系统结构

Mar. 3, 2025

# 主要内容: 比特、字节 和 整型数

- 以比特表示信息
- 比特运算
- 整型数 (Integers)
  - 无符号数和有符号数
  - 无符号数和有符号数的转换
  - 扩展、截断
  - 整数运算: 加、非、乘、移位
  - 总结
- 内存、指针、字符串表示

# 比特/位 (Bits)

- 比特值为 0 或 1
- 通过以不同方式编码/解释一组比特
  - 计算机确定执行的操作（指令）
  - ... 并表示和操作数字、集合、字符串等

# 宾夕法尼亚大学第一台电子计算机



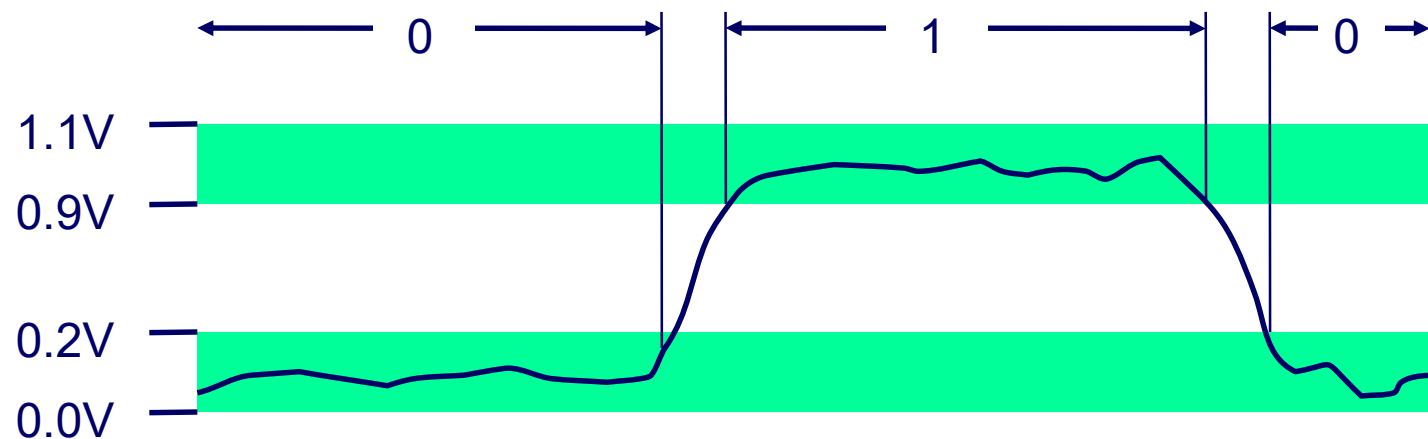
# 中国第一台电子计算机 103



- 研制成功**1958年8月**
- 科研人员在资源匮乏的情况下，凭借坚韧不拔的精神，克服困难，实现了中国计算机领域的突破。
- 当代青年肩负起国家科技创新的责任，为实现中国梦贡献力量。

# 比特/位 (Bits)

- 比特值为 0 或 1
- 通过以不同方式编码/解释一组比特
  - 计算机确定执行的操作（指令）
  - ... 并表示和操作数字、集合、字符串等
- 为什么使用比特？
  - 电气上易于储存，表示
  - 可在嘈杂且不准确的线路上可靠传输



# 例子：二进制计数

## ■ 二进制数表示法

- 将  $15213_{10}$  表示为  $11101101101101_2$
- 将  $1.20_{10}$  表示为  $1.0011001100110011[0011]\dots_2$
- 将  $1.5213 \times 10^4$  表示为  $1.1101101101101_2 \times 2^{13}$

# 字节值编码

## ■ Byte = 8 bits

- 2进制(Binary):  $00000000_2$  to  $11111111_2$
- 10进制(Decimal):  $0_{10}$  to  $255_{10}$
- 16进制(Hexadecimal):  $00_{16}$  to  $FF_{16}$ 
  - 使用字符 '0' 到 '9' 和 'A' 到 'F'
  - $FA1D37B_{16}$  在 C 语言中表示为:
    - `0xFA1D37B`
    - `0xfa1d37b`

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



# 数据表示例子

C Data Type	Typical 32-bit	Typical 64-bit	x86-64
<code>char</code>	1	1	1
<code>short</code>	2	2	2
<code>int</code>	4	4	4
<code>long</code>	4	8	8
<code>float</code>	4	4	4
<code>double</code>	8	8	8
<code>long double</code>	–	–	10/16
<code>pointer</code>	4	8	8

# 主要内容: 比特、字节 和 整型数

- 以比特表示信息
- 比特运算
- 整型数 (Integers)
  - 无符号数和有符号数
  - 无符号数和有符号数的转换
  - 扩展、截断
  - 整数运算: 加、非、乘、移位
  - 总结
- 内存、指针、字符串表示

# 布尔代数 (Boolean Algebra)

- **George Boole** 在19世纪提出
  - 逻辑的代数表示
    - 将“True”编码为 1, “False”编码为 0

# 布尔代数 (Boolean Algebra)

## ■ George Boole 在19世纪提出

### ■ 逻辑的代数表示

- 将“True”编码为 1, “False”编码为 0

### 与 And

- 当A=1 并且 B=1时,  $A \& B = 1$

$\&$	0	1
0	0	0
1	0	1

### 或 Or

- 当A=1 或 B=1时  $A | B = 1$

$ $	0	1
0	0	1
1	1	1

### 非 Not

- 当A=0时,  $\sim A = 1$

$\sim$	
0	1
1	0

### 异或 (Xor)

- 当A=1 或 B=1且两者不同时为1,  $A \wedge B = 1$

$\wedge$	0	1
0	0	1
1	1	0

# 布尔代数 (Boolean Algebra)

## ■ George Boole 在19世纪提出

■ 这

与 And

■ 当A=1

&	
0	
1	

非 Not

克劳德·香农

■ 当A=0时， $\sim A = 1$

$\sim$	
0	1
1	0

■ 当A=1或B=1且两者不同时为1， $A \vee B = 1$

$\vee$	0	1
0	0	1
1	1	0

# 布尔代数 (Boolean Algebra)

- 位向量操作
  - 与 (&), 或 (|), 异或 (^), 非 (~)
  - 按位运算
- 布尔代数的全部性质均适用

$\begin{array}{r} 01101001 \\ \& 01010101 \\ \hline 01000001 \end{array}$	$\begin{array}{r} 01101001 \\   01010101 \\ \hline 01111101 \end{array}$	$\begin{array}{r} 01101001 \\ \wedge 01010101 \\ \hline 00111100 \end{array}$	$\begin{array}{r} 01101001 \\ \sim 01010101 \\ \hline 10101010 \end{array}$
---	--	---	---

# 示例:集合的表示与运算

## ■ 表示

- 宽度  $w$  个比特的向量表示集合  $\{0, \dots, w-1\}$  的子集
- $a_j = 1$  if  $j \in A$ 
  - 01101001     $\{0, 3, 5, 6\}$
  - 76543210

# 示例:集合的表示与运算

## ■ 表示

- 宽度  $w$  个比特的向量表示集合  $\{0, \dots, w-1\}$  的子集
- $a_j = 1$  if  $j \in A$

- 01101001     $\{0, 3, 5, 6\}$

- 76543210

- 01010101     $\{0, 2, 4, 6\}$

- 76543210



# 示例:集合的表示与运算

## ■ 表示

- 宽度  $w$  个比特的向量表示集合  $\{0, \dots, w-1\}$  的子集
- $a_j = 1$  if  $j \in A$

■ 01101001     $\{0, 3, 5, 6\}$

■ 76543210

■ 01010101     $\{0, 2, 4, 6\}$

■ 76543210

## ■ 运算

- |     |      |          |                        |
|-----|------|----------|------------------------|
| ■ & | 交集   | 01000001 | $\{0, 6\}$             |
| ■   | 并集   | 01111101 | $\{0, 2, 3, 4, 5, 6\}$ |
| ■ ^ | 对称差集 | 00111100 | $\{2, 3, 4, 5\}$       |
| ■ ~ | 补集   | 10101010 | $\{1, 3, 5, 7\}$       |

# C语言中的位级运算

- C语言中的位运算： $\&$ （按位与）、 $|$ （按位或）、 $\sim$ （按位取反）、 $\wedge$ （按位异或）
  - 适用于任何整型数据类型：long, int, short, char, unsigned
  - 将操作数视为位向量
  - 将参数按位运算
- 例子(char 类型)
  - $\sim 0x41 \rightarrow 0xBE$ 
    - $\sim 01000001_2 \rightarrow 10111110_2$
  - $\sim 0x00 \rightarrow 0xFF$ 
    - $\sim 00000000_2 \rightarrow 11111111_2$
  - $0x69 \& 0x55 \rightarrow 0x41$ 
    - $01101001_2 \& 01010101_2 \rightarrow 01000001_2$
  - $0x69 | 0x55 \rightarrow 0x7D$ 
    - $01101001_2 | 01010101_2 \rightarrow 01111101_2$

# C语言中的位级运算

- C语言中的位运算:  $\&$ ,  $|$ ,  $\sim$ ,  $\wedge$ 
  - 适用于任何整型数据类型: long, int, short, char, unsigned
  - 将操作数视为位向量
  - 将参数按位运算
- 例子(char 类型)
  - $\sim 0x41 \rightarrow 0xBE$ 
    - $\sim 01000001_2 \rightarrow 10111110_2$
  - $\sim 0x00 \rightarrow 0xFF$ 
    - $\sim 00000000_2 \rightarrow 11111111_2$
  - $0x69 \& 0x55 \rightarrow 0x41$ 
    - $01101001_2 \& 01010101_2 \rightarrow 01000001_2$
  - $0x69 | 0x55 \rightarrow 0x7D$ 
    - $01101001_2 | 01010101_2 \rightarrow 01111101_2$

# C语言的逻辑运算

- C语言中的位运算:  $\&$ ,  $|$ ,  $\sim$ ,  $\wedge$
- C语言的逻辑运算符:  $\&\&$ ,  $||$ ,  $!$ 
  - 将0 视作 逻辑“False(假)”
  - 所有非0值视作逻辑 “True(真)”
  - 计算结果总是0 或 1
  - 提前终止(Early termination)、短路求值(short cut)
- 例子(char 数据类型)
  - $!0x41 \rightarrow 0x00$
  - $!0x00 \rightarrow 0x01$
  - $!!0x41 \rightarrow 0x01$
  - $0x69 \&\& 0x55 \rightarrow 0x01$
  - $0x69 || 0x55 \rightarrow 0x01$

十进制转十六进制:  $65 \div 16 = 4$  余1, 因此高位为4, 低位为1, 组合为**0x41** (ASCII编码中, 大写字母A-Z对应十六进制**0x41-0x5A**)

# C语言中的移位运算

## ■ 左移: $x \ll y$

- 将位向量 $x$ 向左移动  $y$ 位
  - 扔掉左边多出(移出)的位
  - 在右边补0

## ■ 右移: $x \gg y$

- 将位向量 $x$ 向右移动  $y$ 位
  - 扔掉右边多出(移出)的位
- 逻辑右移
  - 在左边补0
- 算术右移
  - 复制左边的最高位

<b>Argument x</b>	01100010
<b><math>\ll 3</math></b>	00010000
<b>Log. <math>\gg 2</math></b>	00011000
<b>Arith. <math>\gg 2</math></b>	00011000

<b>Argument x</b>	10100010
<b><math>\ll 3</math></b>	00010000
<b>Log. <math>\gg 2</math></b>	00101000
<b>Arith. <math>\gg 2</math></b>	11101000

# 未定义行为

- 假设我们有一个8位二进制数
  - 例如：01010101（十进制 85）
- 如果我们尝试将所有8位都左移到最左侧，即左移8位：
  - 01010101 << 8
- 由于8位整数只能存储8位数据，这相当于把所有的比特位都推到范围之外，剩下的值是完全不可预测的，因为：
  - 位丢失：超出位宽的比特会被丢弃，导致数据损坏。
  - 溢出：如果左移后最高位变为1（符号位），可能会导致负数或不可预测的结果。
  - 处理器相关：不同的CPU架构对超出位宽的移位可能会有不同的实现方式。大多数机器上，你会得到原值，编译器会对移位量取模  $n \ll k$  // 等价于  $n \ll (k \% 8)$  // 对于8位数据

# 主要内容: 比特、字节 和 整型数

- 以比特表示信息
- 比特运算
- 整型数 (Integers)
  - 无符号数和有符号数
  - 无符号数和有符号数的转换
  - 扩展、截断
  - 整数运算: 加、非、乘、移位
  - 总结
- 内存、指针、字符串表示



# 整数编码

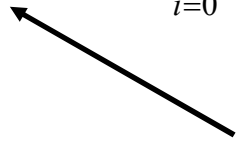
## Unsigned (无符号)

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

## Two's Complement (补码)

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

符号位



## ■ C short:2 bytes long

	Decimal	Hex	Binary
<b>x</b>	15213	3B 6D	00111011 01101101
<b>y</b>	-15213	C4 93	11000100 10010011

## ■ Sign Bit

- 对于补码(2's complement), 最高位表示符号
  - 0 表示非负数
  - 1 表示负数



# 数值范围

## ■ 无符号数值

$$\begin{array}{l} \blacksquare \text{ } UMin = 0 \\ \text{000...0} \end{array}$$

$$\begin{array}{l} \blacksquare \text{ } UMax = 2^w - 1 \\ \text{111...1} \end{array}$$

## ■ 补码数值

$$\begin{array}{l} \blacksquare \text{ } TMin = -2^{w-1} \\ \text{100...0} \end{array}$$

$$\begin{array}{l} \blacksquare \text{ } TMax = 2^{w-1} - 1 \\ \text{011...1} \end{array}$$

## ■ Other Values

$$\begin{array}{l} \blacksquare \text{ Minus 1} \\ \text{111...1} \end{array}$$

位数  $W = 16$

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

# 不同字长的数值

	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

## ■ 观察

- $|TMin| = TMax + 1$ 
  - 非对称
- $UMax = 2 * TMax + 1$

# 无符号数与有符号数编码的值

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

## ■ 相同点

- 非负数值的编码相同

## ■ 唯一性

- 每个位模式对应一个唯一的整数值
- 每个可描述整数有一个唯一编码

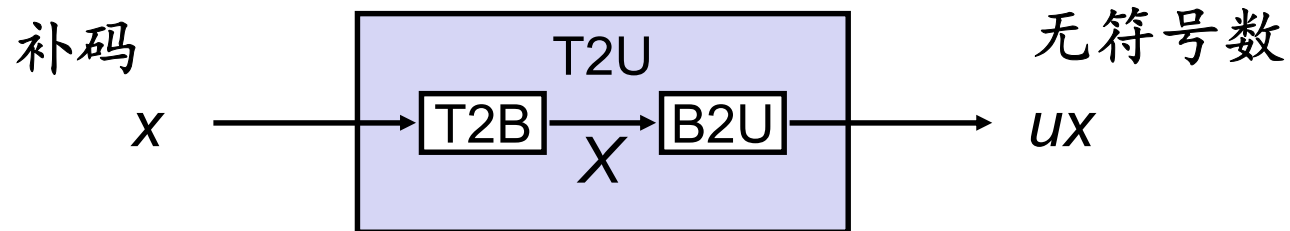
## ■ 可以反转映射关系

- $U2B(x) = B2U^{-1}(x)$ 
  - 无符号整数的位模式
- $T2B(x) = B2T^{-1}(x)$ 
  - 补码的位模式

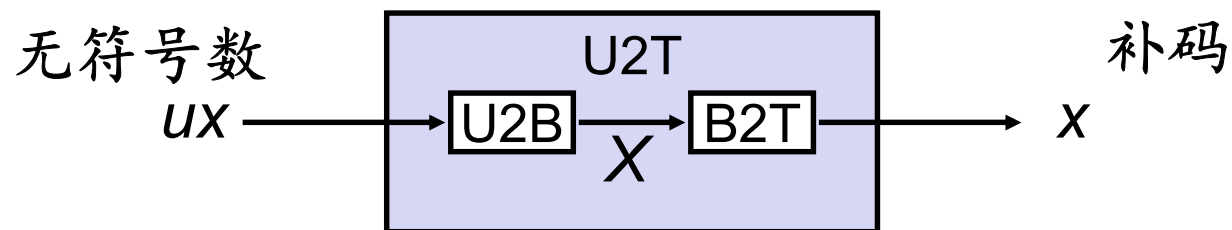
# 主要内容: 比特、字节 和 整型数

- 以比特表示信息
- 比特运算
- 整型数 (Integers)
  - 无符号数和有符号数
  - 无符号数和有符号数的转换
  - 扩展、截断
  - 整数运算: 加、非、乘、移位
  - 总结
- 内存、指针、字符串表示

# 有符号/无符号数之间的转换



位模式相同



位模式相同

## ■ 有符号数和无符号数转换规则:

位模式不变、数值可能改变(按不同编码规则重新解读)

# 有符号无符号数的转换

例如在 5 位补码系统中:

比特模式	补码解释 (signed)	无符号解释 (unsigned)
11111	-1	31
10000	-16	16
00000	0	0
01111	15	15

可以看到:

- **最高位是 0** → 正数 → **补码值 = 无符号值**;
- **最高位是 1** → 负数 → **无符号值 = 补码值 +  $2^w$**  (这里  $w=5$ ) 。

所以, T2U 和 U2T 在正数部分没有差别, 而在负数部分会产生“跳跃”:

$$T2U(-1) = 31$$

$$T2U(-16) = 16$$

# 有符号无符号数的转换

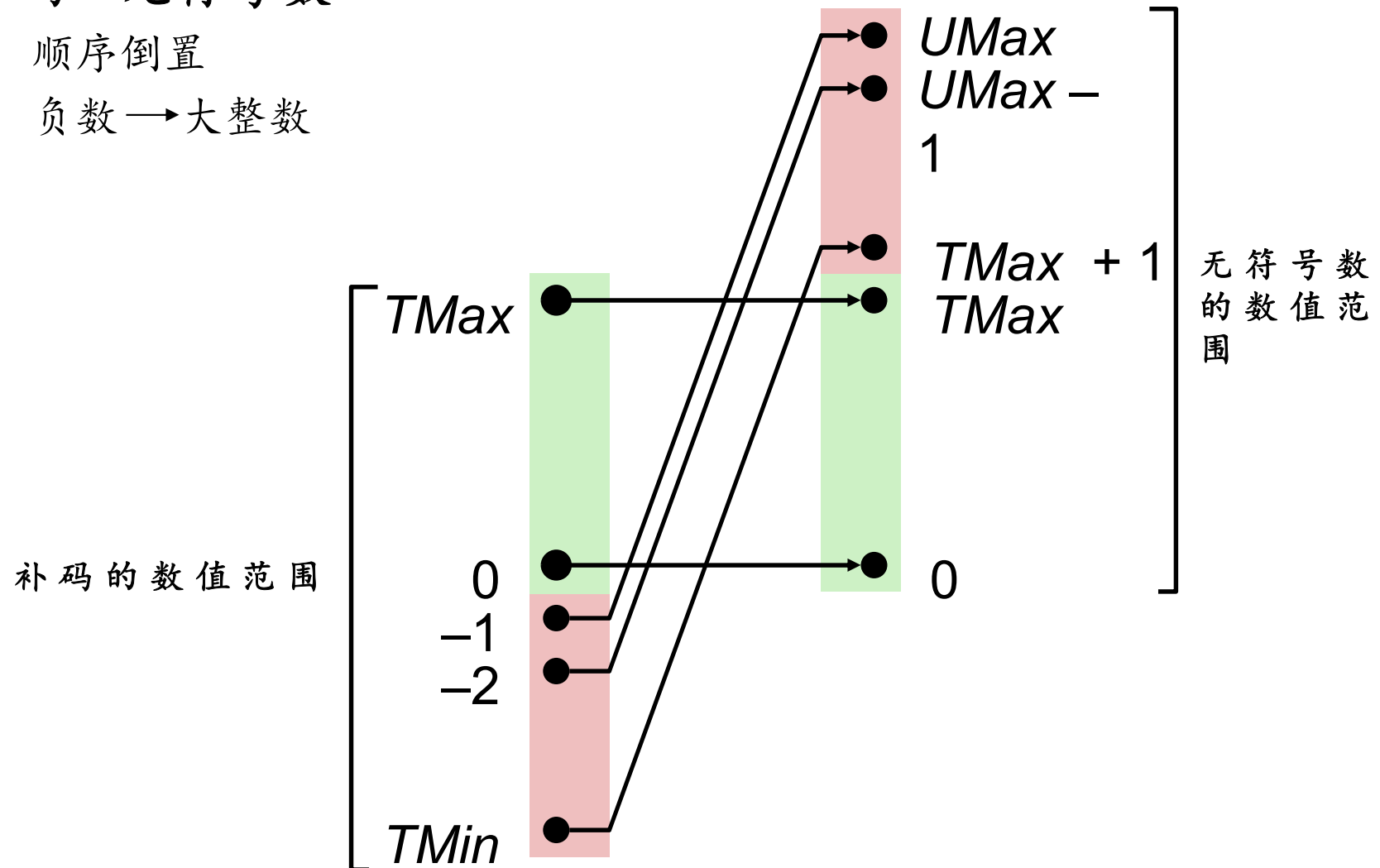
Bits	Signed		Unsigned
0000	0	=	0
0001	1		1
0010	2		2
0011	3		3
0100	4		4
0101	5		5
0110	6		6
0111	7		7
1000	-8	+/- 16	8
1001	-7		9
1010	-6		10
1011	-5		11
1100	-4		12
1101	-3		13
1110	-2		14
1111	-1		15

对于  $w$  位二进制数来说，有符号补码和无符号数的数值差，正好是  $2^w$ 。

# 转换的可视化

## ■ 补码→无符号数

- 顺序倒置
- 负数→大整数





# C语言中的有符号数和无符号数

## ■ 常量

- 数字默认是有符号数
- 无符号数用后缀“U”

**0U, 4294967259U**

## ■ 类型转换

- 显示的强制类型转换

**int tx, ty;**

**unsigned ux, uy;**

**tx = (int) ux;**

**uy = (unsigned) ty;**

- 隐式的类型转换 (赋值、函数调用等情况下发生)

**tx = ux;**

如果两个数都是有符号数 → 按有符号数处理

**uy = ty;**

如果两个数都是无符号数 → 按无符号数处理

如果一个是有符号，另一个是无符号 → 自动把有符号数转换为无符号数

# 类型转换

## ■ 表达式计算

■ 表达式中有符号和无符号数混用时:

■ 有符号数隐式转换为无符号数

■ 包括比较运算符  $<$ ,  $>$ ,  $==$ ,  $<=$ ,  $>=$

■ 例如  $W = 32$ :  **$TMIN = -2,147,483,648$** ,  **$TMAX = 2,147,483,647$**

■ Constant <sub>1</sub>	Constant <sub>2</sub>	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

# 总结：有符号数和无符号数转换的基本原则

- 位模式不变
- 重新解读（按目标编码类型的规则解读）
- 会有意外副作用：数值被 **+ or -  $2^w$**
- 表达式含无符号数和有符号数时
  - 有符号数被转换成无符号数（如int 转成unsigned int）

# 主要内容: 比特、字节 和 整型数

- 以比特表示信息
- 比特运算
- 整型数 (**Integers**)
  - 无符号数和有符号数
  - 无符号数和有符号数的转换
  - 扩展、截断
  - 整数运算: 加、非、乘、移位
  - 总结
- 内存、指针、字符串表示

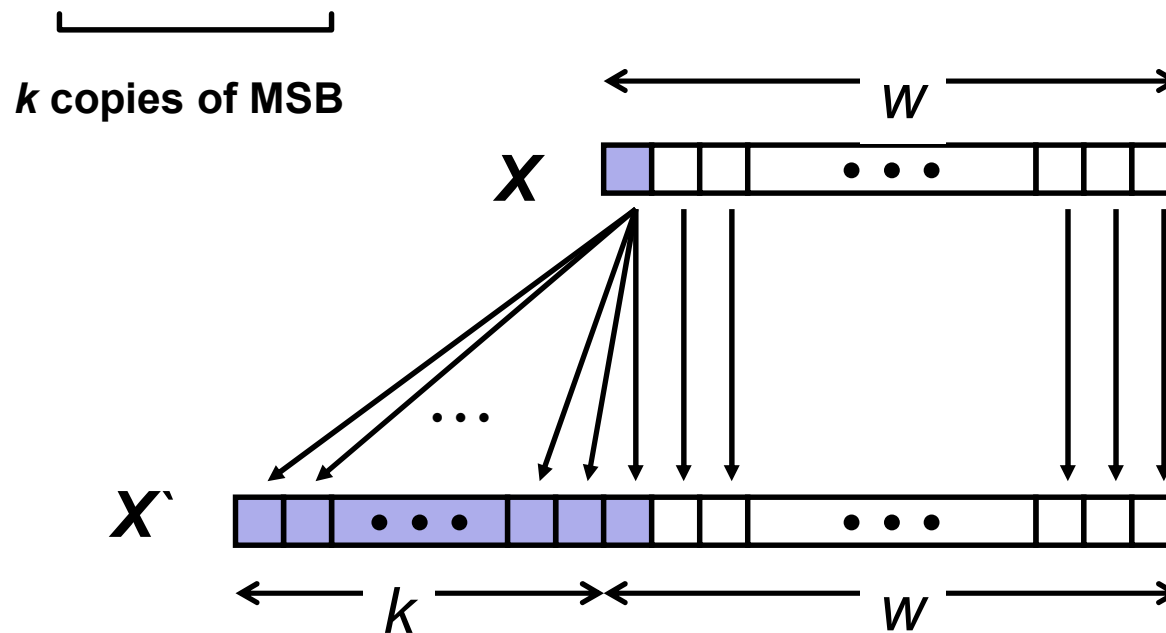
# 符号扩展

## ■ 任务:

- 给定  $w$ -bit 的有符号整型数  $x$
- 将其转换为  $w+k$ -bit 的相同数值的整型数

## ■ 规则:

- 将最高有效位复制  $k$  份:
- $X' = \underbrace{x_{w-1}, \dots, x_{w-1}}_{k \text{ copies of MSB}}, x_{w-1}, x_{w-2}, \dots, x_0$



# 符号扩展示例

```
short int x = 15213;  
int      ix = (int) x;  
short int y = -15213;  
int      iy = (int) y;
```

	Decimal	Hex	Binary
x	15213	3B 6D	00111011 01101101
ix	15213	00 00 3B 6D	00000000 00000000 00111011 01101101
y	-15213	C4 93	11000100 10010011
iy	-15213	FF FF C4 93	11111111 11111111 11000100 10010011

- 从短整数类型向长整数类型转换时，**C**自动进行符号扩展

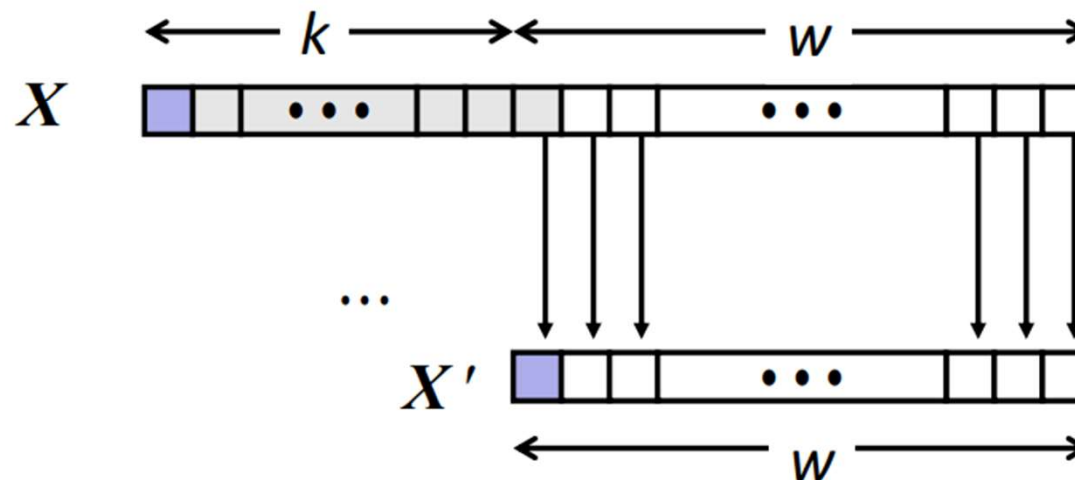
# 截断

## ■ 任务:

- 给定  $k+w$  位的有符号或无符号整数  $X$
- 转换成  $w$  位的整数  $X'$  (与足够小的  $X$  数值相同)

## ■ 规则:

- 丢弃最高的  $k$  位:
- $X' = x_{w-1}, x_{w-2}, \dots, x_0$



# 截断例子

## No sign change

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

	-8	4	2	1
2 =	0	0	1	0

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

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

## Sign change

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

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

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

	-8	4	2	1
6 =	0	1	1	0



# 总结:扩展、截断的基本规则

## ■ 扩展 (例如从**short int** 到**int**的转换)

- 无符号数: 填充0
- 有符号数: 符号扩展
- 结果都是明确的预期值

## ■ 截断 (例如从**unsigned** 到**unsigned short**的转换)

- 无论有/无符号数: 多出的位均被截断
- 结果重新解读
- 无符号数: 相当于求模运算
- 有符号数: 与求模运算相似
- 对于小整数, 结果是明确的预期值

下周

# 主要内容: 比特、字节 和 整型数

- 以比特表示信息
- 比特运算
- 整型数 (Integers)
  - 无符号数和有符号数
  - 无符号数和有符号数的转换
  - 扩展、截断
  - 整数运算: 加、非、乘、移位
  - 总结
- 内存、指针、字符串表示