整数

Integer

本章内容

Topic

- 编码

  Encoding

- 变换

  Conversions

- 运算

  Operations

编码

Encoding

方括号中的内容是可以省略的

Text in square brackets is optional

整数的表示
Integer Representations

C语言数据类型 C data type	Minimum		Maximum	
	32-bit machine	64-bit machine	32-bit machine	64-bit machine
char	-128		127	
unsigned char	0		255	
short [int]	-32,768		32,767	
unsigned short [int]	0		65,535	
int	-2,147,483,648		2,147,483,647	
unsigned int	0		4,294,967,295	
long [int]	-2,147,483,648	-9,223,372,036,854,775,808	2,147,483,647	9,223,372,036,854,775,807
unsigned long [int]	0		4,294,967,295	18,446,744,073,709,551,615
long long [int]	-9,223,372,036,854,775,808		9,223,372,036,854,775,807	
unsigned long long [int]	0		18,446,744,073,709,551,615	

编码

Encoding

$B2U_w$：函数，二进制转无符号数

$B2T_w$：函数，二进制转补码

二进制编码、无符号数和有符号数
Binary, Unsigned & Signed

- 假设一个整数数据类型有w位，其二进制编码可以用一个位向量 $\vec{x}$ 来表示

Consider an integer data type of w bit. We write a bit vector as either $\vec{x}$.

$$\vec{x} = (x_{w-1}, x_{w-2}, x_{w-3}, \ldots\ldots, x_0)$$

Unsigned
无符号数编码的定义

$$B2U_w(\vec{x}) = \sum_{i=0}^{w-1} x_i 2^i$$

Two's Complement
有符号数编码（补码）的定义

$$B2T_w(\vec{x}) = -x_{w-1}2^{w-1} + \sum_{i=0}^{w-2} x_i 2^i$$

Sign Bit
符号位

无符号数的编码
Unsigned Encodings

$$B2U_w(\vec{x}) = \sum_{i=0}^{w-1} x_i 2^i$$

$$B2U_4([0001]) = 0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 0 + 0 + 0 + 1 = 1$$

$$B2U_4([0101]) = 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 0 + 4 + 0 + 1 = 5$$

$$B2U_4([1011]) = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 8 + 0 + 2 + 1 = 11$$

$$B2U_4([1111]) = 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 8 + 4 + 2 + 1 = 15$$

有符号数的编码（补码）
Two's-Complement Encodings

$$B2T_w(\vec{x}) = -x_{w-1}2^{w-1} + \sum_{i=0}^{w-2} x_i 2^i$$

$$B2T_4([0001]) = -0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 0 + 0 + 0 + 1 = 1$$

$$B2T_4([0101]) = -0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 0 + 4 + 0 + 1 = 5$$

$$B2T_4([1011]) = -1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = -8 + 0 + 2 + 1 = -5$$

$$B2T_4([1111]) = -1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = -8 + 4 + 2 + 1 = -1$$

补码示例
Two's-Complement Encoding Examples

- 12345 编码的二进制/十六进制形式

Binary/Hexadecimal Representation for 12345

Binary: 0011 0000 0011 1001

Hex: 3 0 3 9

- -12345 编码的二进制/十六进制形式

Binary/Hexadecimal Representation for –12345

Binary: 1100 1111 1100 0111

Hex: C F C 7

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

编码

Encoding

常见的无符号数有符号数编码
Unsigned & Signed Numeric Values

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

- 等价性　Equivalence
  - 有符号数和无符号数的非负值编码相同

    Same encodings for nonnegative values
- 唯一性　Uniqueness
  - 每个编码都表示唯一的整数值

    Every bit pattern represents unique integer value
  - 每整数有唯一的编码

    Each representable integer has unique bit encoding
- ⇒ 可以反向映射

  ⇒ Can Invert Mappings
  - $U2B(x) = B2U^{-1}(x)$
  - $T2B(x) = B2T^{-1}(x)$

编码

Encoding

将整数转换为补码
Encode Integer to 2's Comp.

- 如果该数字值非负 （If nonnegative），补码等于该值对应的二进制数（位长不足补0）
- 否则 （Otherwise），其绝对值的二进制数逐位取反，并加1，符号位置1
- 举例：假设某个整数类型位长为4

  Example: consider an integer data type has 4-bit width

  - Integer: 5

    Raw binary: 101 （二进制数）

    2's complement : 0101 （补码）

  - Integer: -5

    Absolute value raw binary: 101 (绝对值二进制)

    After complement: 010 （逐位取反）

    2's complement: 1011 （补码）

以上方法仅限应用于可在编码表示范围内的数字

可表示的整数范围
Numeric Range

- 无符号数

  Unsigned Values

  - $U_{min} = 0$
  - $U_{max} = 2^{w}-1$

- 有符号数（补码）

  Two's Complement Values

  - $T_{min} = -2^{w-1}$
  - $T_{max} = 2^{w-1}-1$

一些重要的数字
Important numbers

	Word size w			
Value	8	16	32	64
$UMax_w$	0xFF	0xFFFF	0xFFFFFFFF	0xFFFFFFFFFFFFFFFF
	255	65,535	4,294,967,295	18,446,744,073,709,551,615
$TMin_w$	0x80	0x8000	0x80000000	0x8000000000000000
	−128	−32,768	−2,147,483,648	−9,223,372,036,854,775,808
$TMax_w$	0x7F	0x7FFF	0x7FFFFFFF	0x7FFFFFFFFFFFFFFF
	127	32,767	2,147,483,647	9,223,372,036,854,775,807
−1	0xFF	0xFFFF	0xFFFFFFFF	0xFFFFFFFFFFFFFFFF
0	0x00	0x0000	0x00000000	0x0000000000000000

可表示的整数范围
Numeric Range

- 关系 Relationship
  - $|T_{Min}| = T_{Max} + 1$
  - $U_{max} = 2 \times T_{Max} + 1$
- 有符号数-1的补码和U_{max}的编码相同

  -1 has the same bit representation as U_{max}
  - 所有的位都为 1

    a string of all 1s
- 0 的编码方式都相同，编码所有的位都为0

  Numeric value 0 is represented as a string of all 0s in both representations

其他的编码方式
Alternative Representations

- 反码 One's Complement
  - 和补码的定义类似，区别是符号位的权重为 $-(2^{w-1}-1)$

    The most significant bit has weight $-(2^{w-1}-1)$
- 原码 Sign-Magnitude
  - 和补码的区别是，符号位的作用仅用于决定其他位的位权为正/负

    The most significant bit is a sign bit that determines whether the remaining bits should be given negative or positive weight

本章内容

Topic

- 编码

  Encoding

- 变换

  Conversions

  - 有符号数与无符号数

    Signed vs. unsigned

  - 扩展与截断

    Extension and Truncation

- 运算

  Operations

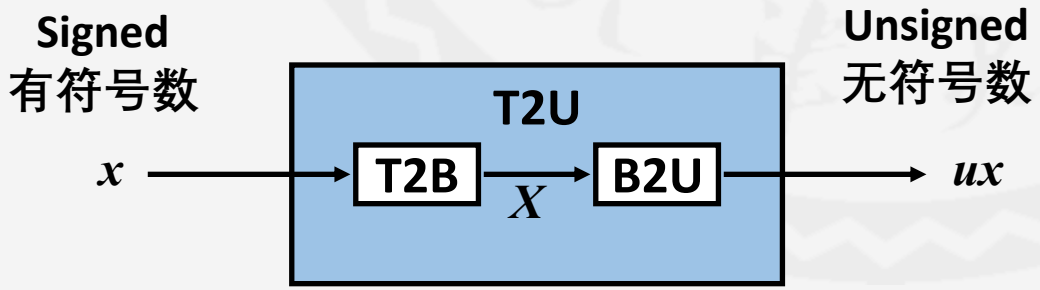
变换

Conversions

有符号数转无符号数
Casting Signed to Unsigned

- C语言允许将有符号数转换为无符号数
  C Allows Conversions from Signed to Unsigned
- 转换结果
  Resulting Value
  - 编码本身没有发生变化
    No change in bit representation
  - 非负数没有发生变化
    Nonnegative values unchanged
    - ux = 12345
- 负数转换为一个（大）正数
  Negative values change into (large) positive values
  - uy = 53191

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



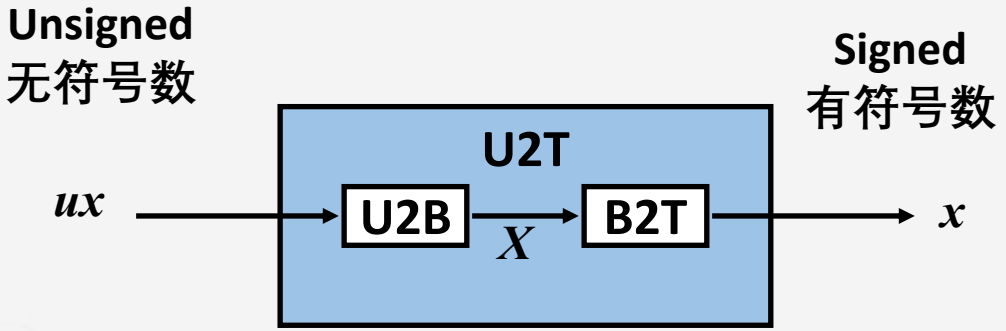
Maintain Same Bit Pattern
编码保持不变

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

变换

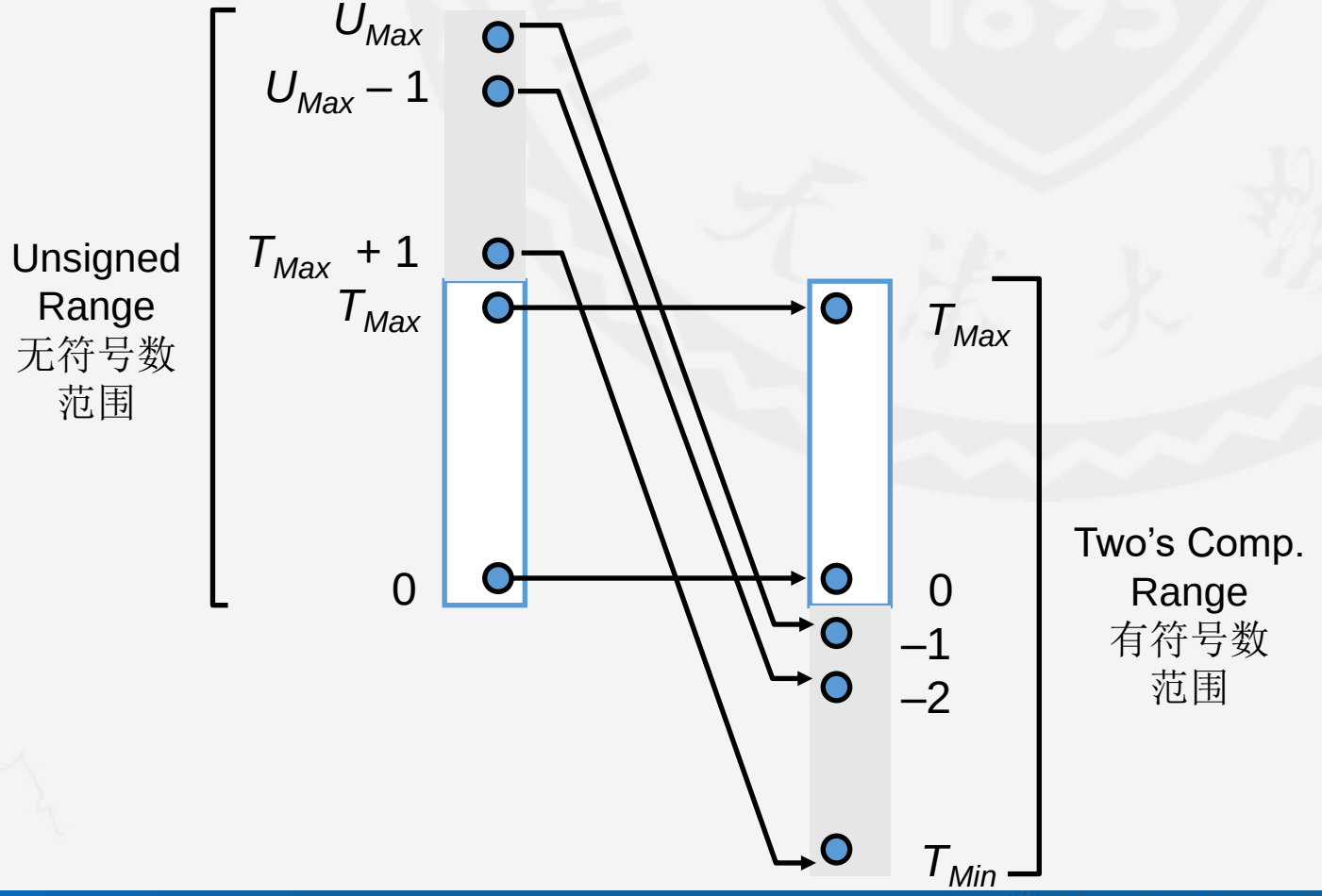
Conversions

无符号数转有符号数
Casting Unsigned to Signed



Maintain Same Bit Pattern
编码保持不变

$$x = \begin{cases} ux & ux \le T_{Max} \\ ux - 2^w & ux > T_{Max} \end{cases}$$



变换

Conversions

C语言中的有符号数和无符号数
Signed vs. Unsigned in C

- 常量

  Constant

  - 缺省情况下，所有的整数常量都是有符号数

    By default are considered to be signed integers

  - 如果需要声明无符号数常量，需要增加一个后缀 “U”

    Unsigned if have “U” as suffix

    - 0U, 4294967259U

变换

Conversions

C语言中的有符号数和无符号数之间的转换
Casting Between Signed and Unsigned in C

显式转换

Explicit casting

```
int tx, ty;
unsigned ux, uy;
tx = (int) ux;
uy = (unsigned) ty;
```

隐式转换

Implicit casting

```
int tx, ty;
unsigned ux, uy;
tx = ux;                /* Cast to signed */
uy = ty;                /* Cast to unsigned */
```

隐式转换时，编译器会产生Warning
(不推荐)

变换

Conversions

C语言中的表达式求值
Expression Evaluation in C

- 如果在一个表达式中混用无符号数和有符号数

  If mix unsigned and signed in single expression

  - 有符号数会被隐式转换成无符号数

    Signed values implicitly cast to unsigned

- 比较运算也采用上述规则 <, >, ==, <=, >=

  Including comparison operations <, >, ==, <=, >=

举例：表达式求值
Example: Expression Evaluation

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

以32位字长为例
Examples for word size = 32

本章内容

Topic

- 编码

  Encoding

- 变换

  Conversions

  - 有符号数与无符号数

    Signed vs. unsigned

  - 扩展与截断

    Extension and Truncation

- 运算

  Operations

位扩展
Expanding the Bit Representation

- 无符号数：零扩展

  Unsigned: Zero extension

  - 扩展后最高位的空位补0

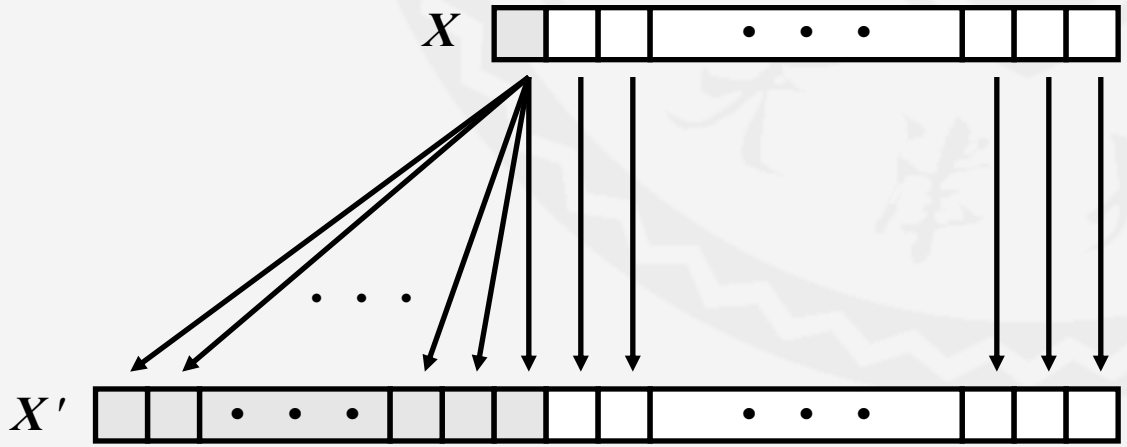
    Add leading 0s to the representation

- 有符号数：符号位扩展

  Signed: Sign extension

  - 扩展后最高位的空位原编码的符号位

    Add copies of the most significant bit to the representation



示例：符号位扩展
Sign Extension Example

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

	Decimal	Hex	Binary
x	12345	30 39	00110000 00111001
ix	12345	00 00 30 39	00000000 00000000 00110000 00111001
y	-12345	CF C7	11001111 11000111
iy	-12345	FF FF CF C7	11111111 11111111 11001111 11000111

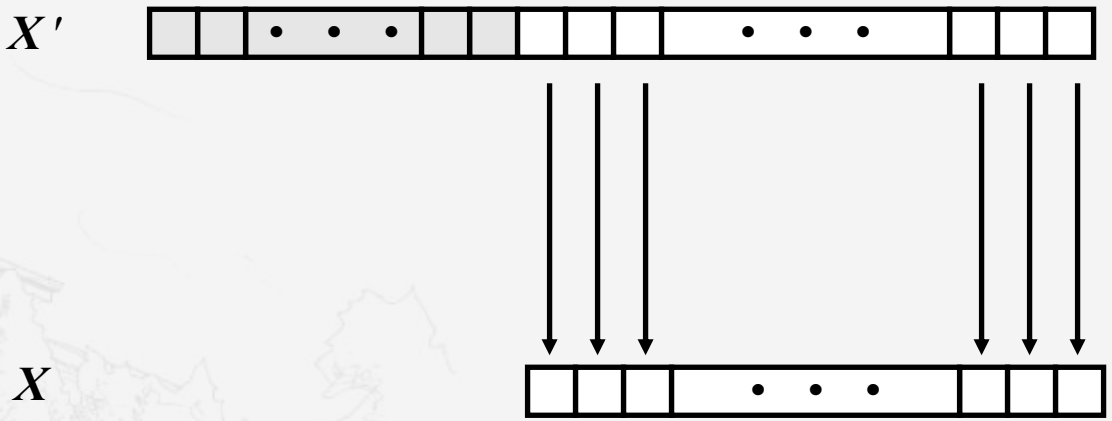
数字截断
Truncating Numbers

```
int   x  = 53191;
short sx = (short) x;
int   y  = sx;
```

	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

多余的位被直接丢弃

Discard the leading bits



变换

Conversions

小知识：什么情况下可以使用无符号数?
Why Should I Use Unsigned?

- 不能仅因为参数值非负就使用无符号数

  Don't Use Just Because Number Nonnegative

  - 很容易导致逻辑错误

    Easy to make mistakes

```
unsigned i;
for (i = cnt-2; i >= 0; i --)
  a[i] += a[i-1]
```

  - 有些错误很难发现

    Can be very subtle

```
#define DELTA sizeof(int)
int i;
for (i = cnt; i-DELTA >= 0; i -= DELTA)
  ...
```

- 可以使用的情况

  Do Use When

  - 位向量

    Bit Vectors
  - 掩码

    Mask
  - 地址 （内存地址、网络地址）

    Address
  - 高精度计算

    Multiprecision Arithmetic

思考题

```
/* WARNING: This is buggy code */
float sum_elements(float a[], unsigned length) {
  int i;
  float result = 0;

  for (i = 0; i <= length-1; i++)
    result += a[i];
  return result;
}
```

```
/* stdio.h */
typedef unsigned int  size_t;

/* Prototype library function strlen */
size_t strlen(const char *s);

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

思考题

```
/* Declaration of library function memcpy */
void *memcpy(void *dest, void *src, size_t n);

/* Kernel memory region holding user-accessible data */
#define KSIZE 1024
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);
    return len;
}
```

本章内容

Topic

- 编码
  Encoding
- 变换
  Conversions
- 运算
  Operations
  - 加法
    Addition
  - 有符号数的相反数
    Two's-Complement Negation
  - 乘法
    Multiplication
  - 使用移位运算实现2的幂的乘法和除法
    Using Shift to perform power-of-2 multiply/divide

运算

Operations

数学上的整数加法
Mathematical Integer Addition

- 整数加法

  Integer Addition

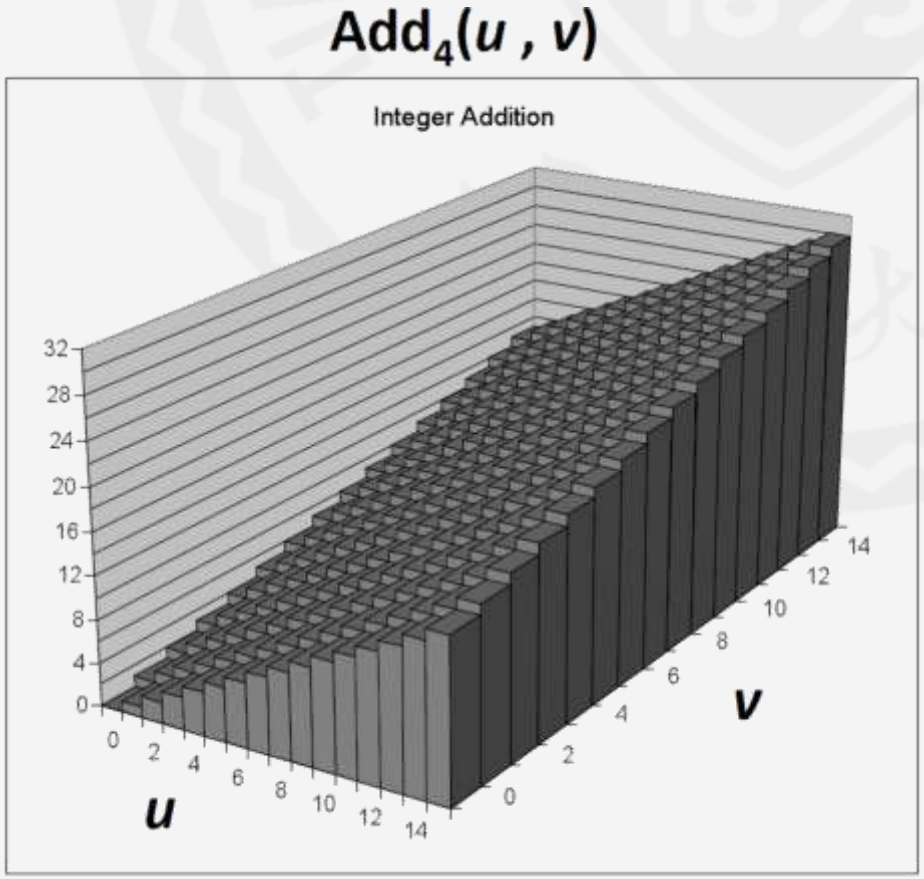
  - 4比特的两个整数 u、v

    4-bit integers u,v
  - $Add_4(u, v)$函数用来计算数学上的和

    Compute true sum Add4(u, v)
  - 计算结果随着u和v的增长呈现线性的变化

    Values increase linearly with u and v
  - 在图像上形成了一个平面

    Form planar surface



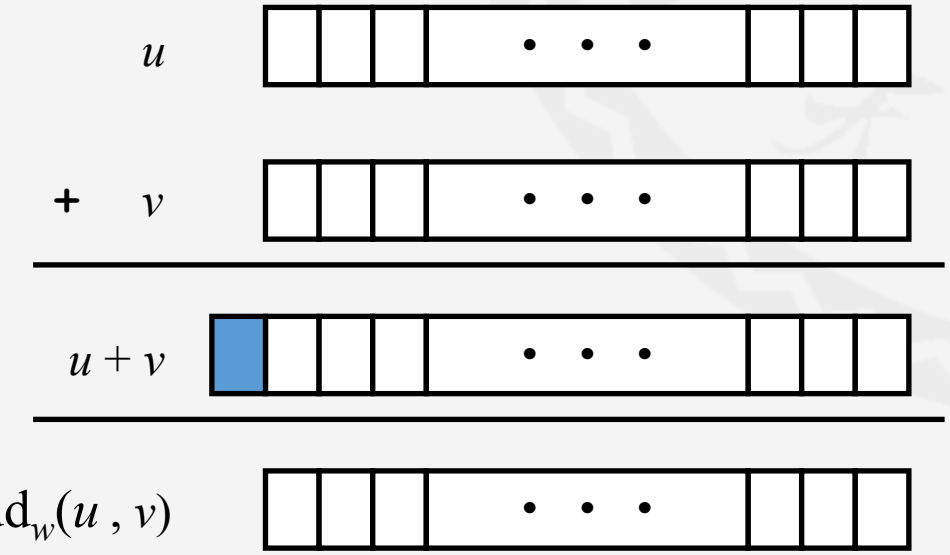
无符号数加法规则
Unsigned Addition

Operands: *w* bits
操作数：w位

True Sum: *w*+1 bits
真实和：w+1位

Discard Carry: *w* bits
丢弃进位位后：w位

$\mathrm{UAdd}_w(u, v)$



无符号数加法规则
Unsigned Addition

- 编码按照二进制加法规则相加，忽略进位位

  Standard Addition Function, and then ignores carry output

- 丢弃进位位等价于进行一次取模运算

  Implements Modular Arithmetic

  $s = \mathrm{UAdd}_w(u, v) = \mathrm{ADD}_w(u, v) \mod 2^w$

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

无符号数加法的图形表示
Unsigned AdditionVisualizing Unsigned Addition

- 无符号数的和出现了截断

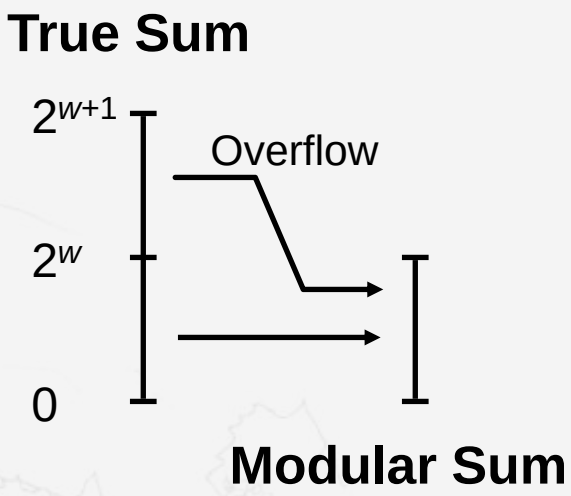
  Wraps Around

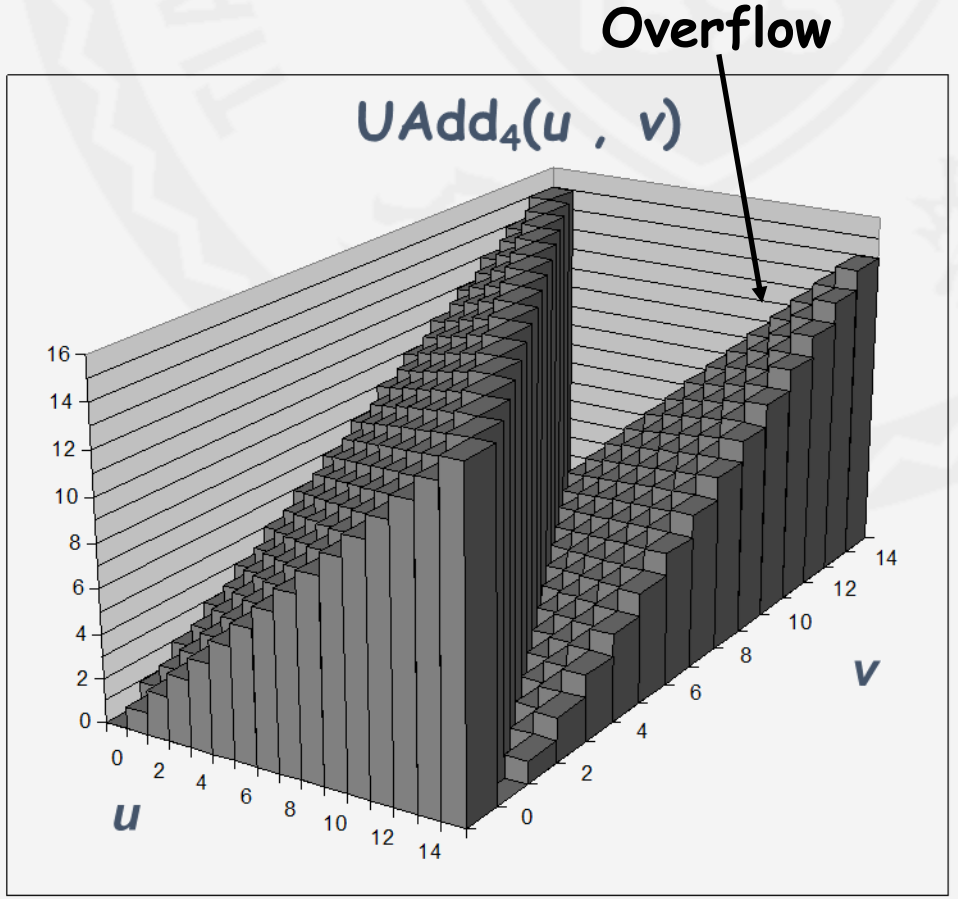
  - 当真实和大于等于2^{w}时

    If true sum ⩾ 2^{w}

  - 只会有一次截断

    At most once





运算

Operations

阿贝尔群 （可交换群、加群）
Abelian Group

1. 具有封闭的加法运算 Closed under addition

   $0 \le \text{UAdd}_w(u, v) \le 2^w - 1$

2. 交换律 Commutative

   $\text{UAdd}_w(u, v) = \text{UAdd}_w(v, u)$

3. 结合律 Associative

   $\text{UAdd}_w(t, \text{UAdd}_w(u,v)) = \text{UAdd}_w(\text{UAdd}_w(t, u), v)$

4. 具有唯一的0元素 0 is additive identity

   $\text{UAdd}_w(u, 0) = u$

5. 每个元素都有补元素 Every element has additive inverse

   Let $\text{UComp}_w(u) = 2^w - u$

   $\text{UAdd}_w(u, \text{UComp}_w(u)) = 0$

有符号数加法
Signed Addition

$$TAdd(u,v) = U2T\left(UAdd\big(T2U(u), T2U(v)\big)\right)$$

- 规则

Functionality

- 真实和共w+1位

  True sum requires w+1 bits

- 丢弃最高位

  Drop off MSB

- 将计算结果视为补码编码

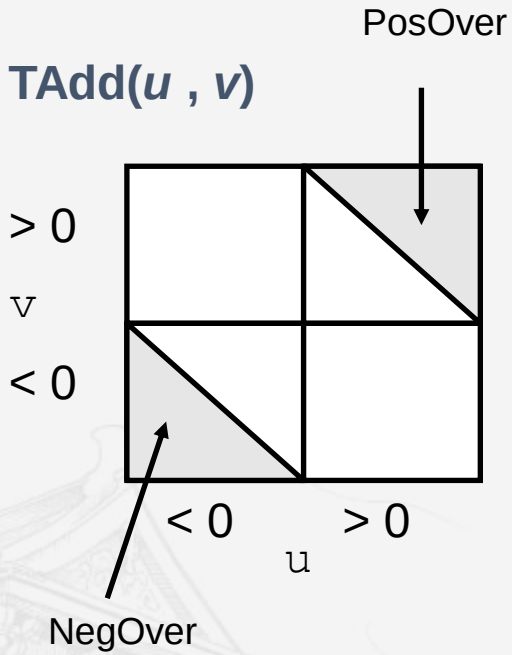
  Treat remaining bits as 2's comp. integer

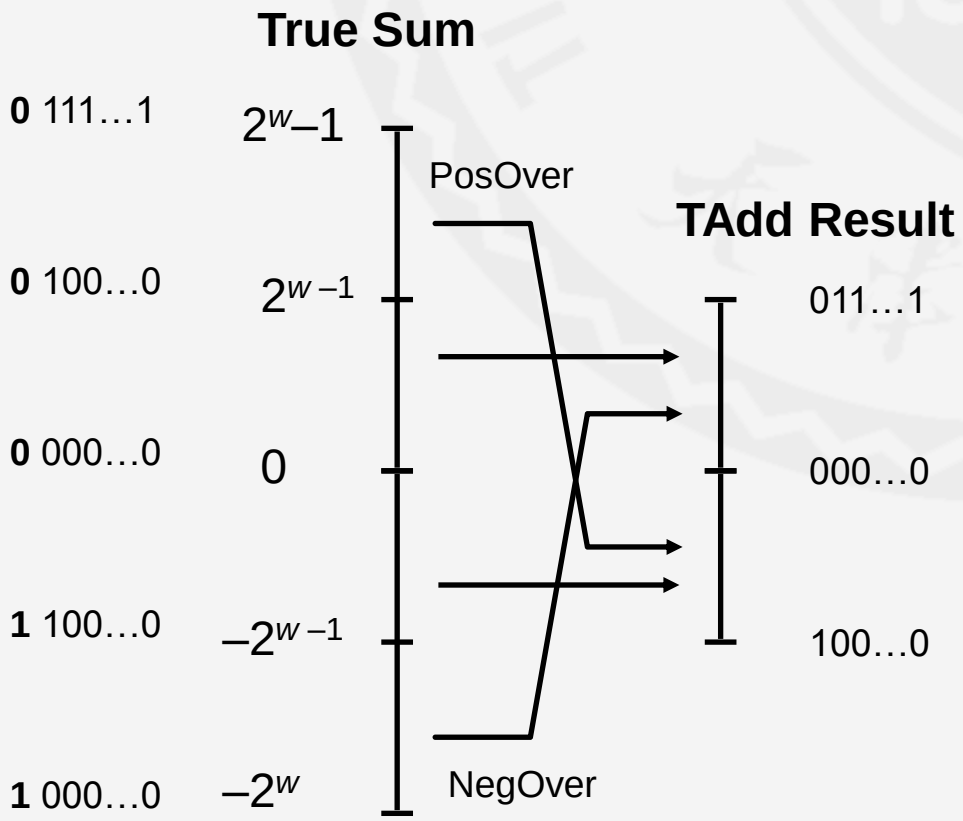
$$TAdd(u,v) = \begin{cases} u+v-2^w & u+v > TMax \quad (PosOver) \\ u+v & TMin \le u+v \le TMax \\ u+v+2^w & u+v < TMin \quad (NegOver) \end{cases}$$

PosOver: Positive Overflow 正溢出

NegOver: Negative Overflow 负溢出

有符号数加法
Signed Addition





运算

Operations

有符号数加法的图形表示
Visualizing Signed Addition

- 4比特补码运算

  4-bit two's comp.

- 值的范围 -8 ~ +7

  Range from -8 to +7

- 和截断

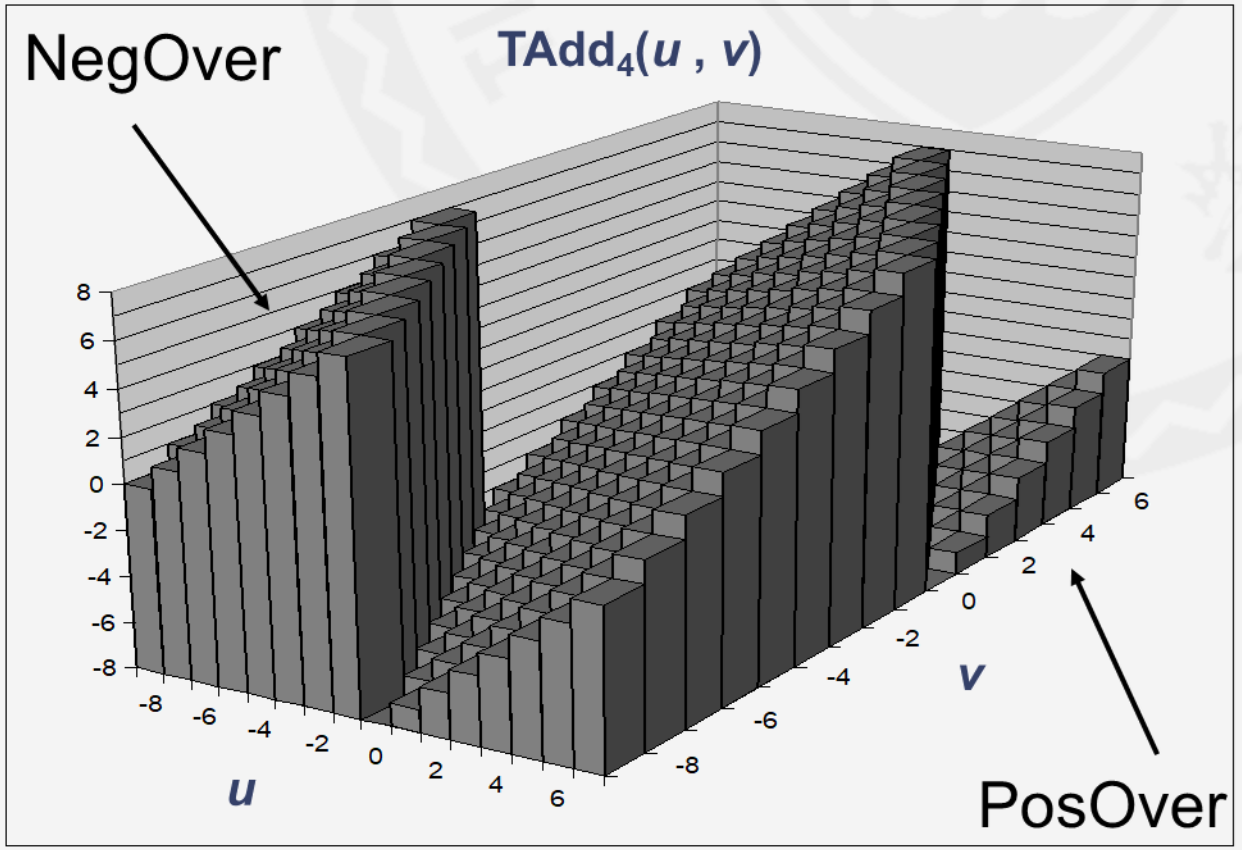
  Wraps Around

  - 如果大于等于2^{w-1}，变为负数

    If sum ≥ 2^{w-1}, becomes negative

  - 如果小于-2^{w-1}，变为正数

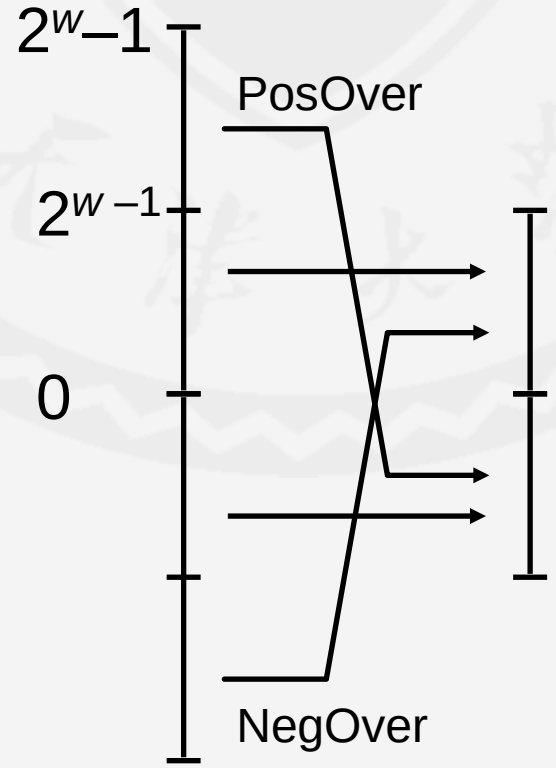
    If sum < -2^{w-1}, becomes positive



有符号数加法溢出问题讨论
Detecting TAdd Overflow

- 设 $s = \text{TAdd}_w(u, v)$，
- 当且仅当以下情况均不出现时：
  - $u, v < 0, s \geq 0$ (负溢出)
  - $u, v \geq 0, s < 0$ (正溢出)
- $s == \text{Add}_w(u, v)$

- 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)



overflow = (u<0 == v<0) && (u<0 != s<0)

本章内容

Topic

- 编码
  Encoding
- 变换
  Conversions
- 运算
  Operations
  - 加法
    Addition
  - 有符号数的相反数
    Two's-Complement Negation
  - 乘法
    Multiplication
  - 使用移位运算实现2的幂的乘法和除法
    Using Shift to perform power-of-2 multiply/divide

使用取反和加法运算求相反数
Negating with Complement & Increment

- 在C语言中有
  In C Language
  - ~x + 1 == -x
- 显然：~x + x == 1111…111 == -1
  Observation: ~x + x == 1111…111 == -1

	x	1	0	0	1	1	1	0	1
+	~x	0	1	1	0	0	0	1	0
	-1	1	1	1	1	1	1	1	1

$$-x = \begin{cases} -2^{w-1} & x = -2^{w-1} \\ -x & x > -2^{w-1} \end{cases}$$

本章内容

Topic

- 编码
  Encoding
- 变换
  Conversions
- 运算
  Operations
  - 加法
    Addition
  - 有符号数的相反数
    Two's-Complement Negation
  - 乘法
    Multiplication
  - 使用移位运算实现2的幂的乘法和除法
    Using Shift to perform power-of-2 multiply/divide

乘法
Multiplication

- 计算两个w位宽整数在数学上的乘积
  Computing Exact Product（积） of w-bit numbers x, y
  - 无论有符号数还是无符号数
    Either signed or unsigned
- 乘积的大小
  Ranges
  - 无符号数：$0 \leqslant x * y \leqslant (2^w - 1)^2 = 2^{2w} - 2^{w+1} + 1$
    Unsigned:$0 \leqslant x * y \leqslant (2^w - 1)^2 = 2^{2w} - 2^{w+1} + 1$
    - 最多2w位
      Up to 2w bits
  - 有符号数最小值：$x*y \geqslant -2^{w-1}*(2^{w-1}-1) = -2^{2w-2} + 2^{w-1}$
    Two's complement min: $x*y \geqslant -2^{w-1}*(2^{w-1}-1) = -2^{2w-2} + 2^{w-1}$
    - 最多2w-1 位 （包括符号位）
      Up to 2w–1 bits (including sign bit)
  - 有符号数最大值：$x * y \leqslant (-2^{w-1})^2 = 2^{2w-2}$
    Two's complement max: $x * y \leqslant (-2^{w-1})^2 = 2^{2w-2}$
    - 最多2w位 （包括符号位）,仅当T_{Min}^2
      Up to 2w bits (including sign bit) but only for TMin2

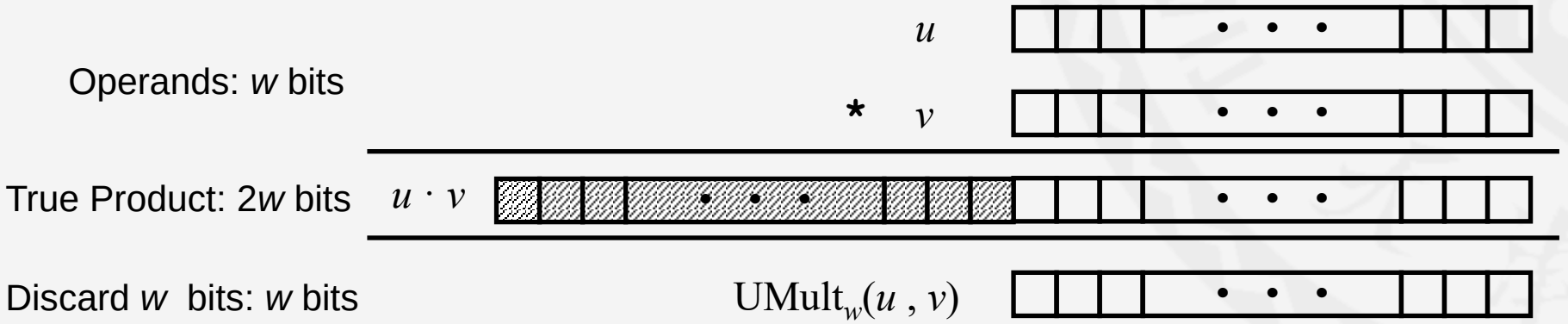
乘法
Multiplication

- 想要获得精确的乘法结果
  Maintaining Exact Results
  - 需要使用额外的存储空间保存扩展出的位宽
    Would need to keep expanding word size with each product computed
  - 在一些高精度数学计算库中通常采用这种方法
    Done in software by “arbitrary precision” arithmetic packages

运算

Operations

无符号数乘法
Unsigned Multiplication



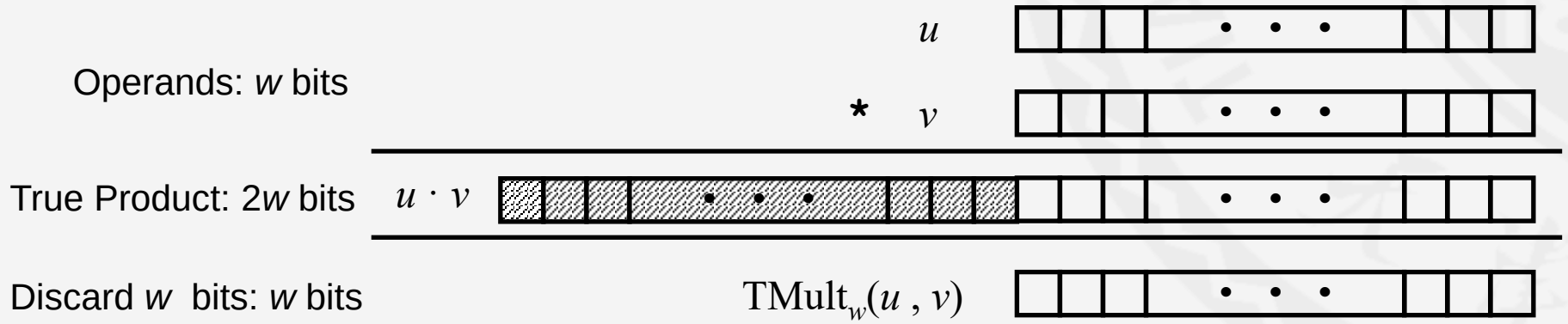
- C语言中的乘法运算
  Multiplication Function in C
  - 忽略掉最高的w位

    Ignores high order w bits
- 等价的模运算实现
  Implements Modular Arithmetic
  - $\text{UMult}_w(u, v) = (u \cdot v) \bmod 2^w$

运算

Operations

有符号数乘法
Signed Multiplication



- C语言中的乘法运算
  Multiplication Function in C
  - 忽略掉高w位
    Ignores high order w bits
  - 有符号数和无符号数乘法在某些地方是有区别的
    Some of which are different for signed vs. unsigned multiplication
  - **但是低位部分总是相同的**
    Lower bits are the same

- 等价的模运算实现
  Implements Modular Arithmetic
  - $TMult_w(u, v) = U2T_w((u \cdot v) \bmod 2^w)$
- 有符号数和无符号数乘法使用相同的指令进行运算
  The same instruction is used for signed & unsigned multiplication

本章内容

Topic

- 编码
  Encoding
- 变换
  Conversions
- 运算
  Operations
  - 加法
    Addition
  - 有符号数的相反数
    Two's-Complement Negation
  - 乘法
    Multiplication
  - 使用移位运算实现2的幂的乘法和除法
    Using Shift to perform power-of-2 multiply/divide

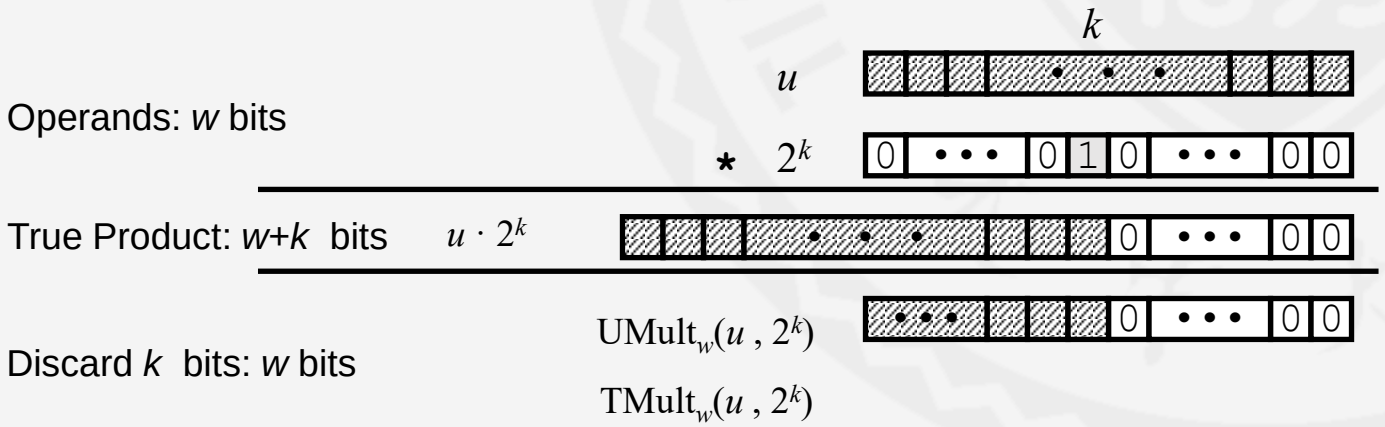
运算

Operations

使用移位实现乘以2的幂
Power-of-2 Multiply with Shift

- 运算
  Operation
  - u << k 等价于 u * 2^k
    u << k gives u * 2^k
  - 同时适用于有符号数和无符号数
    Both signed and unsigned

- 例如
  Examples
  - (u << 3) == (u * 8)
  - (u << 5) - (u << 3) == u * 24



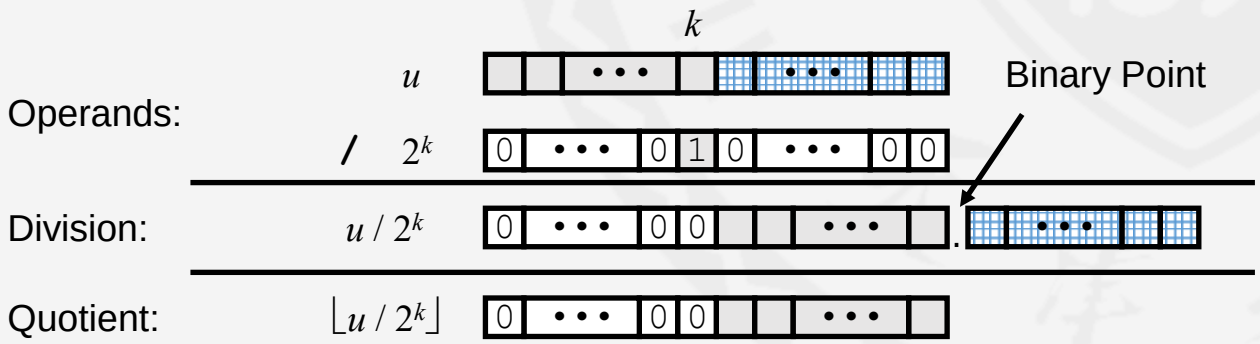
- 绝大多数计算机上移位运算和加法运算比乘法运算快得多
  Most machines shift and add much faster than multiply
  - 编译器会自动生成这种模式的代码
    Compiler will generate this code automatically

运算

Operations

使用移位实现无符号数除以2的幂
Unsigned Power-of-2 Divide with Shift

- 无符号数除以2的幂得到的商
  Quotient of Unsigned by Power of 2
  - u >> k 等价于⌊ u / 2^k ⌋
    u >> k gives⌊ u / 2^k ⌋
  - 使用逻辑右移
    Uses logical shift



	Division	Computed	Hex	Binary
x	15213	15213	3B 6D	00111011 01101101
x >> 1	7606.5	7606	1D B6	00011101 10110110
x >> 4	950.8125	950	03 B6	00000011 10110110
x >> 8	59.4257813	59	00 3B	00000000 00111011

运算

Operations

使用移位实现有符号数除以2的幂
Signed Power-of-2 Divide with Shift

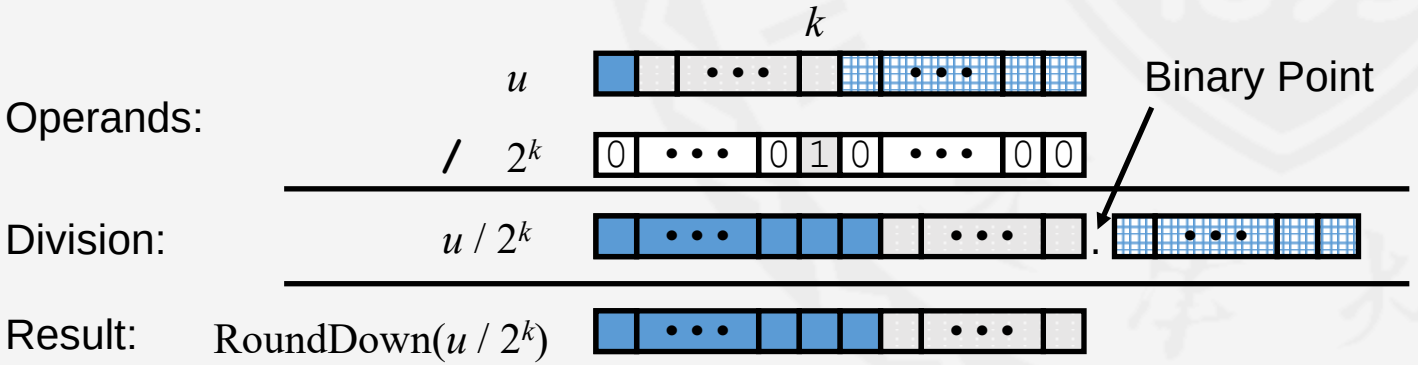
- 有符号数除以2的幂得到的商

  Quotient of Unsigned by Power of 2

  - u >> k 等价于⌊ u / 2^k ⌋

    u >> k gives⌊ u / 2^k ⌋
  - 使用算术右移

    Uses arithmetic shift



- 当u<0时会出现取整方向的错误（C语言整数除法舍入规则是向0取整）

  Rounds wrong direction when u < 0 (Rounding to zero in C)

	Division	Computed	Hex	Binary
y	-15213	-15213	C4 93	11000100 10010011
y >> 1	-7606.5	-7607	E2 49	11100010 01001001
y >> 4	-950.8125	-951	FC 49	11111100 01001001
y >> 8	-59.4257813	-60	FF C4	11111111 11000100

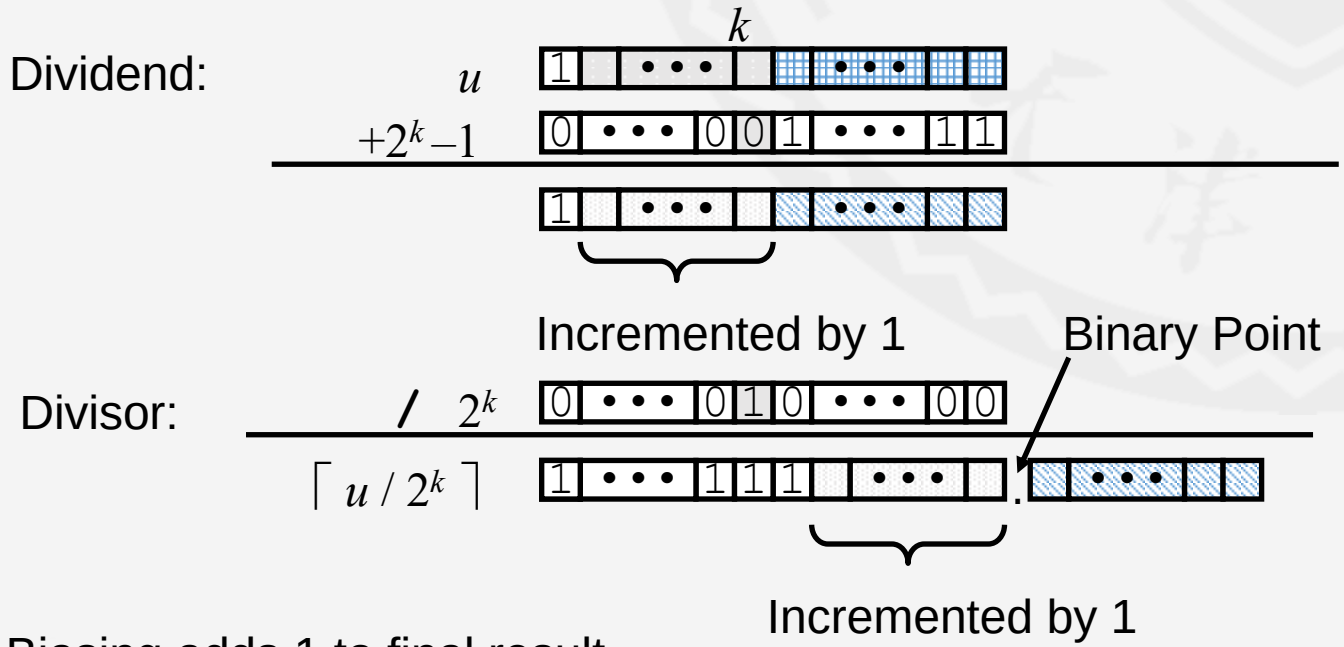
修正除以2的幂的结果
Correct Power-of-2 Divide

- 当u<0时

  when u < 0

  - 为 u 加上2^k-1

    Add 2^k-1 to u



思考：

当u恰好能被整除时会出现问题吗?

Biasing adds 1 to final result

相当于最终结果加1

运算

Operations

小结 Summary

- 加法 Addition
  - 无/有符号数：编码的二进制加法，然后按位长截断
    Unsigned/signed: Normal addition followed by truncate
  - 位运算规则是相同的
    Same operation on bit level
- 乘法 Multiplication
  - 无/有符号数：编码的二进制乘法，然后按位长截断
    Unsigned/signed: Normal multiplication followed by truncate
  - 位运算规则是相同的
    Same operation on bit level

- 相反数 Negation
  - 按位取反然后加1
    Complement and increment 1
- 乘以2的幂 power-of-2 multiply
  - u << k 等价于 $u * 2^k$
    u << k gives $u * 2^k$
- 除以2的幂 power-of-2 divide
  - u >> k 等价于 $\lfloor u * 2^k \rfloor$
    u >> k gives $\lfloor u / 2^k \rfloor$
  - 无符号数：逻辑右移
    Unsigned: logical shift
  - 有符号数：算术右移
    Signed: arithmetic shift
  - 有符号数：小于0时，先加2^k-1
    Signed: when less than 0, add 2^k-1 first