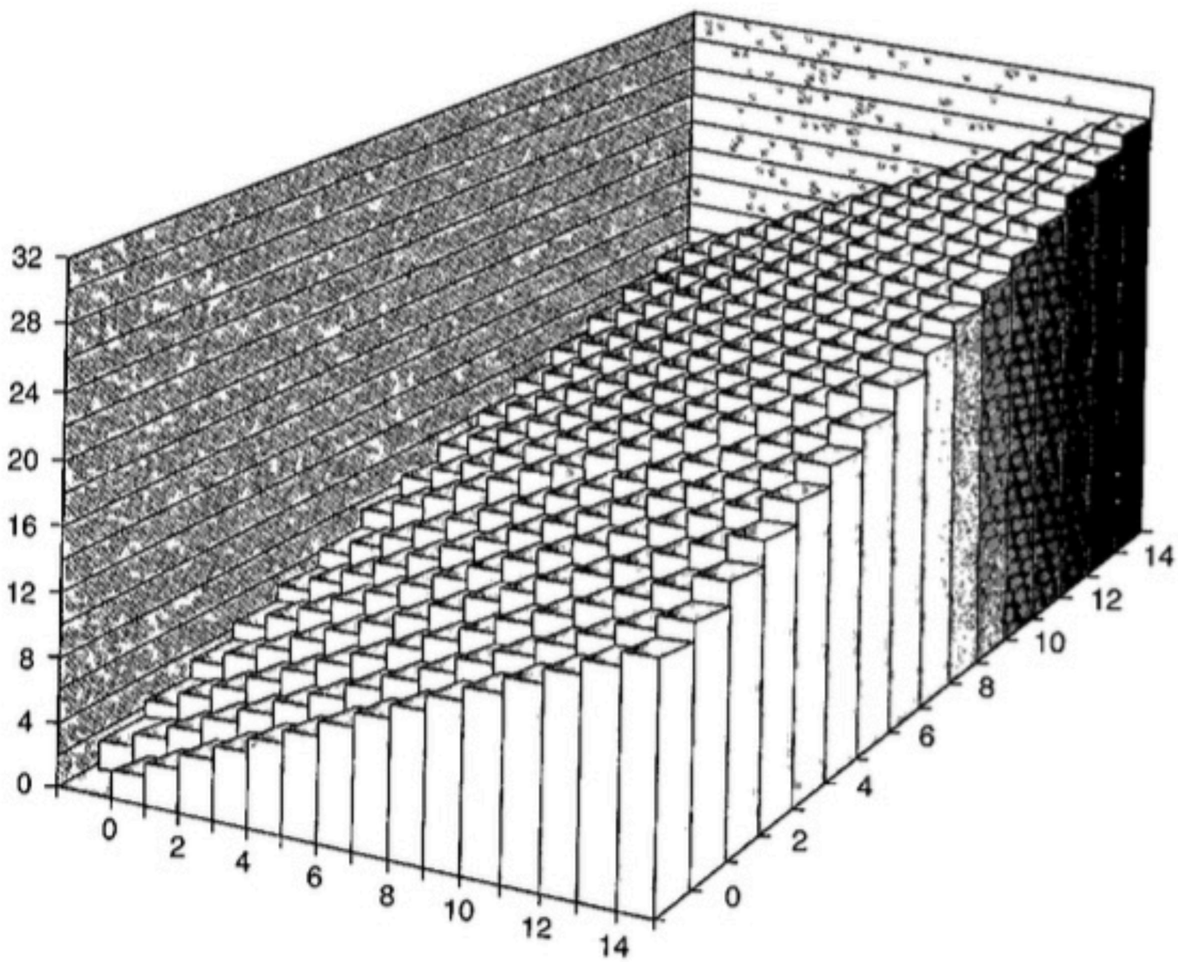
Unsigned Addition

- The sum of two nonnegative integers x and y such that
    - 0 <= x, y < 2^w
- may require w+1 bits to represent.

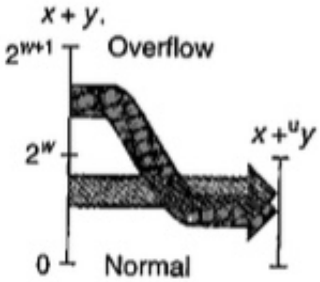


Principle: Unsigned Addition

For x and y such that $0 \leq x,\ y < 2^w$:

$$x +^{u}_{w} y = \begin{cases} x + y, & x + y < 2^w \quad \text{Normal} \\ x + y - 2^w, & 2^w \leq x + y < 2^{w+1} \quad \text{Overflow} \end{cases} \tag{2.11}$$

- The normal case preserves the value of **x + y**

- The overflow case has the effect of decrementing this sum by 2^w.
  - In the case of overflow, the sum will be mod-ed by 2^w
- 

Principle: Detecting Overflow of Unsigned Addition

- The computation of a sum **s = x + y** has overflowed
  - if and only if s < x (or s < y)

```
// Practice Problem 2.27

// Determine whether arguments can be added without overflow.
// Should return 1 if arguments x and y can be added without causing overflow.

int uadd_ok (unsigned x, unsigned y) {
    return ((unsigned) x + y) >= x;
}
```

Principle: Usigned Negation

For any number x such that $0 \leq x < 2^w$, its w-bit unsigned negation $-^{u}_{w}\, x$ is given by the following:

$$-^{u}_{w}\, x = \begin{cases} x, & x = 0 \\ 2^w - x, & x > 0 \end{cases} \tag{2.12}$$

Two’s Complement Addition

Principle: Two’s Complement Addition

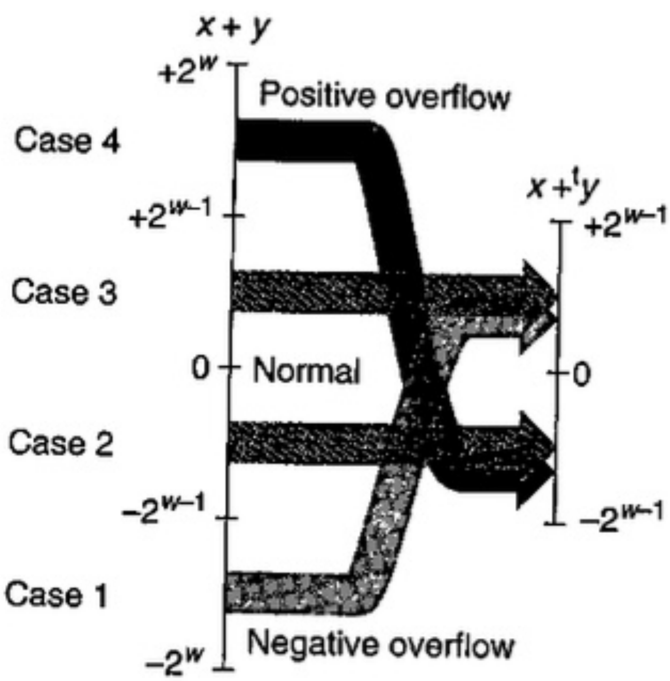
For integer values x and y in the range $-2^{w-1} \leq x, y \leq 2^{w-1} - 1$:

$$x +^t_w y = \begin{cases} x + y - 2^w, & 2^{w-1} \leq x + y \quad \text{Positive overflow} \\ x + y, & -2^{w-1} \leq x + y < 2^{w-1} \quad \text{Normal} \\ x + y + 2^w, & x + y < -2^{w-1} \quad \text{Negative overflow} \end{cases} \tag{2.13}$$

- if addition goes over and causes extra bit to be required, truncate the newest bit
- this has different effects in different cases

Cases:

1. x+y is less than TMin: **negative overflow**
    a. effect of truncation is to add 2^W to the sum
2. x+y is between TMin and 0: **normal**
3. x+y is between 0 and TMax: **normal**
4. x+y exceeds TMax: **positive overflow**
    a. effect of truncation is to subtract 2^W from the sum



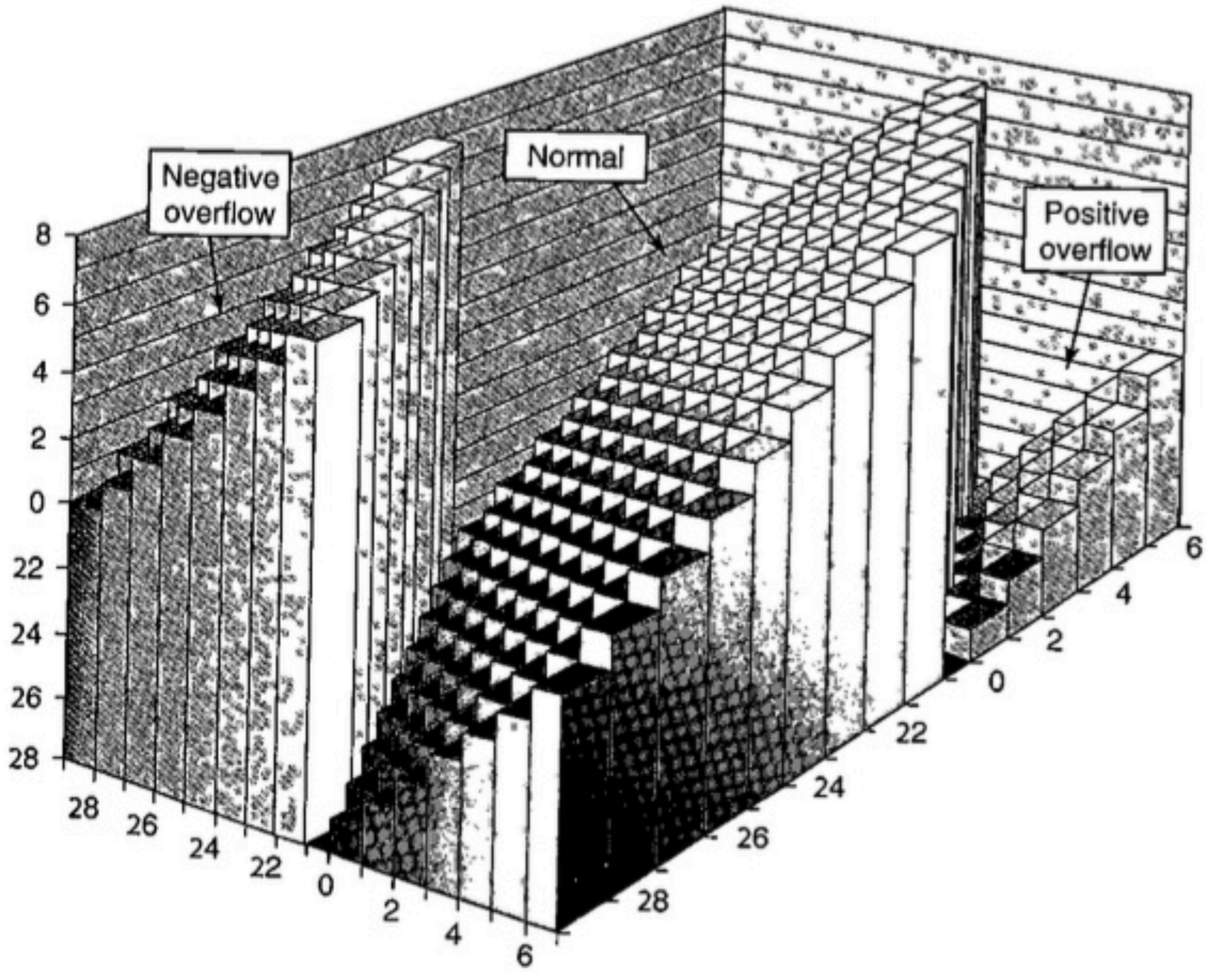
- the w-bit two's-complement sum of two numbers has the exact same bit-level representation as the unsigned sum
    - most computers use the same machine instruction to perform either usigned or signed addition

Two's Complement Addition Examples

x	y	$x+y$	$x +^t_4 y$	Case
-8	-5	-13	3	1
[1000]	[1011]	[10011]	[0011]	
-8	-8	-16	0	1
[1000]	[1000]	[10000]	[0000]	
-8	5	-3	-3	2
[1000]	[0101]	[11101]	[1101]	
2	5	7	7	3
[0010]	[0101]	[00111]	[0111]	
5	5	10	-6	4
[0101]	[0101]	[01010]	[1010]	

Principle: Detecting Overflow in Two's Complement Addition

- For values of x and y from TMin to TMax, inclusive:
  - let s = x + y
- The computation of s had positive overflow IFF
  - x > 0 AND y > 0 BUT s <= 0
- The computation of s had negative overflow IFF
  - x < 0 AND y < 0 BUT s >= 0



With a 4-bit word size, addition can have a negative overflow when x+y < -8 and a positive overflow when x + y >= 8

```c
// Practice Problem 2.30

// Determine whether arguments can be added without overflow.
// Should return 1 if arguments x and y can be added without causing overflow.

int tadd_ok (int x, int y) {
    int result = x + y;
    int neg_over = x < 0 && y < 0 && result >= 0;
    int pos_over = x > 0 && y > 0 && result <= 0;
    return !neg_over && !pos_over;
}
```

Principle: Two’s Complement Negation

For x in the range $TMin_w \leq x \leq TMax_w$, its two's-complement negation $-^t_w\, x$ is given by the formula

$$-^t_w\, x = \begin{cases} TMin_w, & x = TMin_w \\ -x, & x > TMin_w \end{cases} \tag{2.15}$$

Two’s Complement Multiplication

- The result of multiplying w-bit x and y together could require as many as 2w bits to represent in two’s-complement form.
- Signed multiplication is gnerally performed in C by truncating the left w bits from the 2w-bit result.

Principle: Two’s Complement Multiplication

For x and y such that $TMin_w \le x, y \le TMax_w$:

$$x *_w^t y = U2T_w((x \cdot y) \bmod 2^w) \tag{2.17}$$