# CS 261
# Fall 2017

Mike Lam, Professor


## Binary Arithmetic

# Binary Arithmetic

- Topics
  - Basic addition
  - Overflow
  - Multiplication & Division

# Basic addition

- Binary and hex addition are fundamentally the same as decimal addition
  - Add digit-by-digit, using a carry as necessary
  - Result generally requires more bits than the two operands

**Dec**

```
 12540
+ 4683
```

**Bin**

```
 10011100
+ 1010110
```

**Hex**

```
  b0994f
+   7120
```

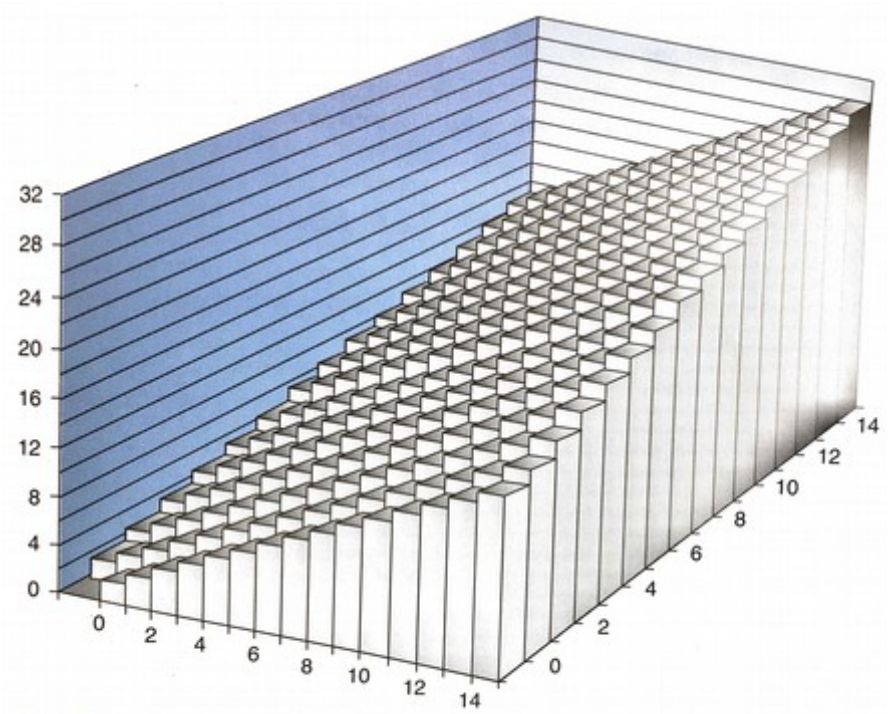



Figure 2.21 **Integer addition.** With a 4-bit word size, the sum could require 5 bits.

# Basic addition

- Binary and hex addition are fundamentally the same as decimal addition
  - Add digit-by-digit, using a carry as necessary
  - Result generally requires more bits than the two operands

```
  11          Dec          111          Bin
  12540                    10011100
+  4683                  +  1010110
 -------                  --------
  17223                    11110010
```

```
   1
  b0994f      Hex
+  7120
 -------
  b10a6f
```

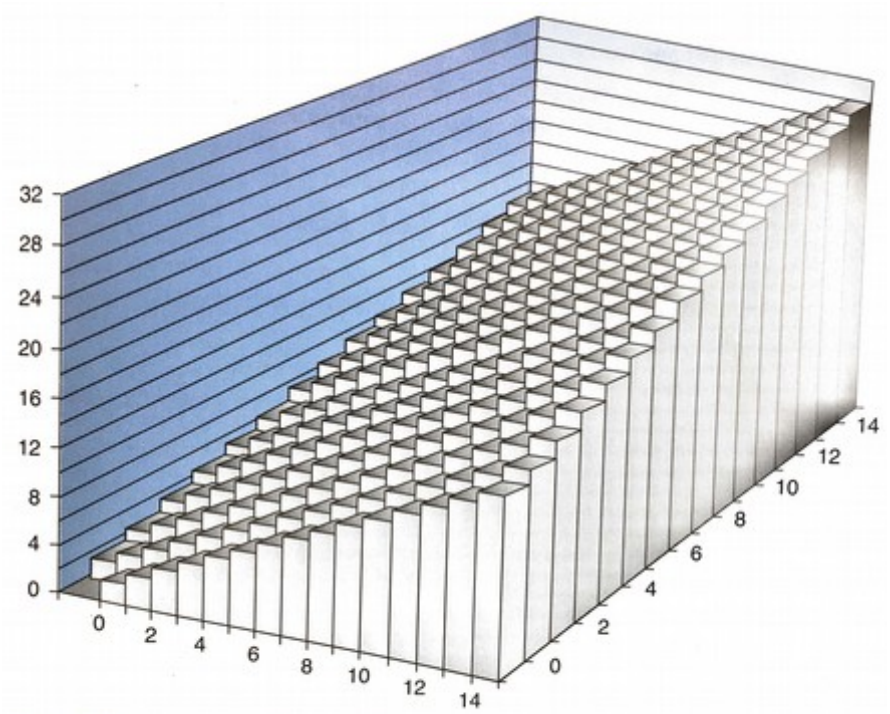



Figure 2.21 **Integer addition.** With a 4-bit word size, the sum could require 5 bits.

# Overflow

- Unsigned addition is subject to overflow
  - Caused by truncation to integer size

```
  1
    994f
+   7120
  -------
  10a6f = 0a6f
```

Truncation!

(assume a 16-bit integer)

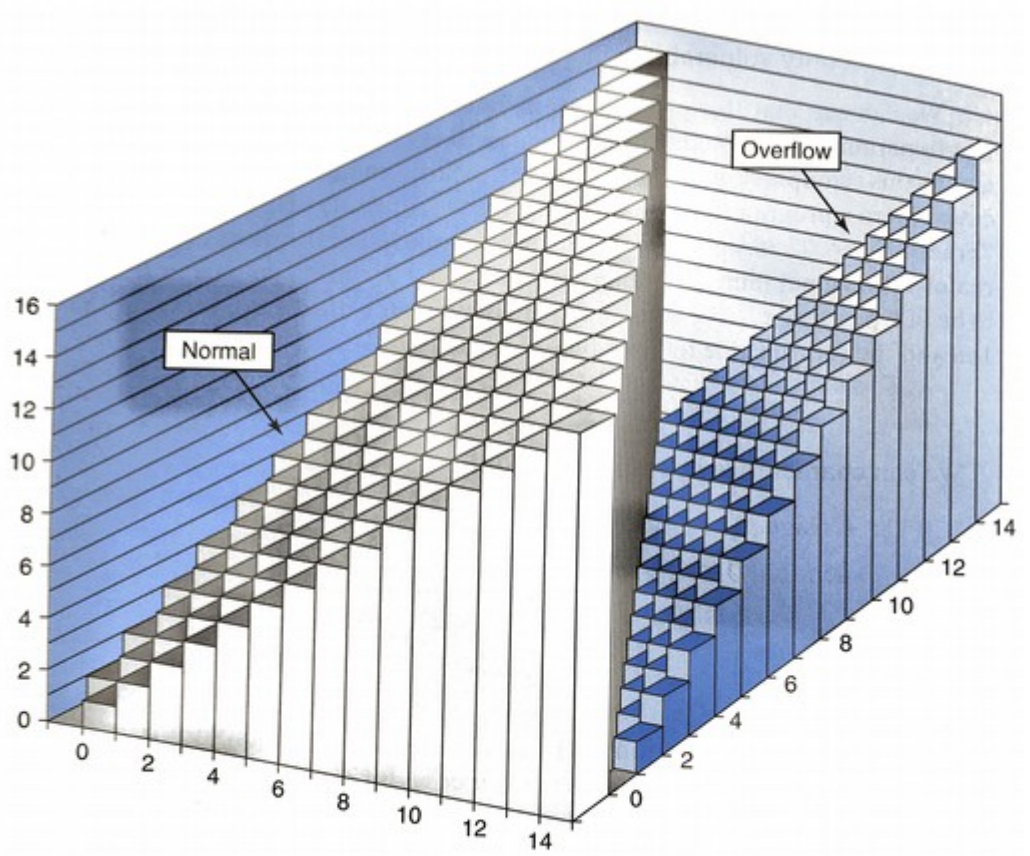
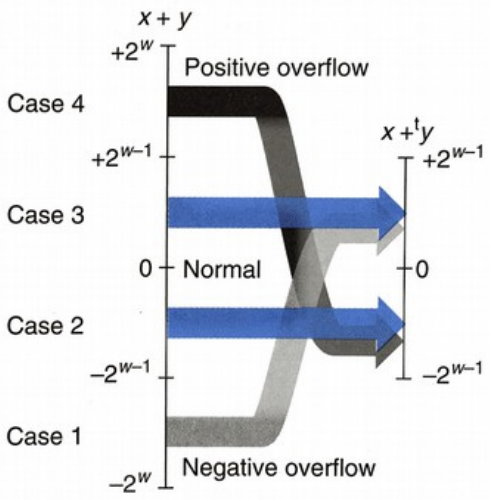



**Figure 2.23 Unsigned addition.** With a 4-bit word size, addition is performed modulo 16.

# Overflow

- Two's complement addition is identical to unsigned mechanically
  - Subject to both positive and negative overflow
  - Overflows if carry-in and carry-out differ for sign bit

**Figure 2.24**
**Relation between integer and two's-complement addition.** When $x + y$ is less than $-2^{w-1}$, there is a negative overflow. When it is greater than or equal to $2^{w-1}$, there is a positive overflow.




Example range (`int8_t`):

```
0111 1111    127
0111 1110    126
…
0000 0001    1
0000 0000    0
1111 1111   -1
…
1000 0001   -127
1000 0000   -128
```

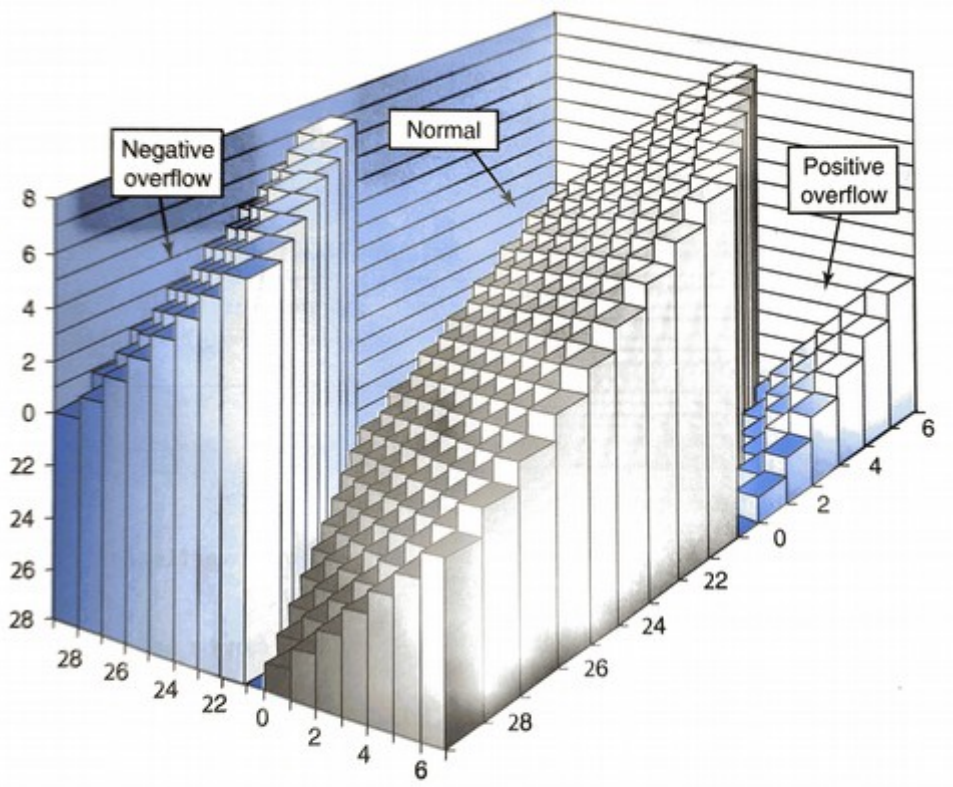



**Figure 2.26** **Two's-complement addition.** With a 4-bit word size, addition can have a negative overflow when $x + y < -8$ and a positive overflow when $x + y \geq 8$.

# Multiplication & Division

- Like addition, fundamentally the same as base 10
  - Actually, it's even simpler!
  - Same regardless of encoding
- Special case: multiply by powers of 2 (shift left)

```
 101 (5)
x 11 (3)
 101
101
1111 (15)
```

```
2 << 1 = 4        (2 * 2)
1 << 2 = 4        (1 * 2 * 2)

1 << 4 = 16       (1 * 2 * 2 * 2 * 2)
4 << 1 = 8        (4 * 2)
4 << 2 = 16       (4 * 2 * 2)
```

- Division is expensive!
  - Special case: divide by powers of two (shift right)

# Binary fractions

- Now we can store integers
  - But what about general real numbers?
- Extend positional binary integers to store fractions
  - Designate a certain number of bits for the fractional part
  - These bits represent negative powers of two
  - (Just like fractional digits in decimal fractions!)

$$101.101$$

4 2 1 1/2 1/4 1/8

$$4 + 1 + 0.5 + 0.125 = \mathbf{5.625}$$

*(alternatively: 5 + 5/8)*

# Another problem

- For scientific applications, we want to be able to store a wide *range* of values
  - From the scale of galaxies down to the scale of atoms
- Doing this with fixed-precision numbers is difficult
  - Even signed 64-bit integers
    - Perhaps allocate half for whole number, half for fraction
    - Range: ~2 x $10^{-9}$ through ~2 x $10^{9}$

# Floating-point numbers

- Scientific notation to the rescue!
  - Traditionally, we write large (or small) numbers as $x \cdot 10^e$
  - This is how floating-point representations work
    - Store exponent and fractional parts (the significand) separately
    - The decimal point “floats” on the number line
    - Position of point is based on the exponent
    - Many nuances and caveats!

$$
\begin{aligned}
& 0.0123 \times 10^2 \\
& 0.123 \times 10^1 \\
1.23 \;=\; & 1.23 \times 10^0 \\
& 12.3 \times 10^{-1} \\
& 123.0 \times 10^{-2}
\end{aligned}
$$