Chapter 3: Number Systems

- Common number systems: Decimal, Binary, Octal, Hexidecimal
- General number system (Base n)
- Real numbers (non-integers)
- Binary Fractions

In the Beginning

```
(Applies to Chapter 4 as well.)

In the beginning people represented numbers with: /
      ///////  is seven
      /////// + ///// = ////////////

We use:
      7 + 5 = 12 = 1*10^1 + 2^0
      -- base ten -- because we have ten fingers
```

BIG CONCEPT (ignored in life but important for computers):

Number representation vs. Number

There are many ways to "represent" a number (e.g., above)

Representation does not affect the result of an operation

```
XII + XXXIII = XLV
12 + 33 = 45
```

But representation affects difficulty of computing result:

```
 XXXIII  (33)
*   XII  (12)
 ------
XXXIII
  XXXIII
 CCCXXX
 --------
 CCCXXXXXXXXXIIIIII
 CCCLXXXXVI
 CCCXCVI = 396
```

For computer we need to chose a representations that allows us to build fast electronic circuits for computer (e.g., adding)

Since computer don't have fingers, they don't use base ten number

Summary:

We will review number systems
The next chapter applies the ideas to computers.

Representing positive integers: 1, 2, 3, ...

Big advance of humankind: Arabic numerals
"weighted position notation with base ten"

```
345 is really
       3 x 100   + 4 x 10    + 5 x 1
       3 x 10**2  + 4 x 10**1  + 5 x 10**0
```

3 is the most significant symbol (it carries the most weight)

5 is the least significant symbol (it carries the least weight)

```
     digits (or symbols) allowed: 0-9
     base (or radix):  10
```

Pronunciation: 'rA-diks

For computer binary number work great
-- why? two-state devices

Binary number system

base (radix)	2
digits allowed	0 1

each binary digit is called a BIT

the order of the digits is significant

```
numbering of the digits
      msb          lsb
      n-1           0
   where n is the number of digits in the number

     msb stands for most significant bit
     lsb stands for least significant bit
```

1001 (base 2) is really
1 x 2**3 + 0 x 2**2 + 0 x 2**1 + 1 x 2**0
9 (base 10)

11000 (base 2) is really
1 x 2**4 + 1 x 2**3 + 0 x 2**2 + 0 x 2**1 + 0 x 2**0
24 (base 10)

	20th C		
system	e.g.	humans	computers
roman	XVIII	bad	bad
Arabic	18	good	bad
binary	100010	bad	good

Humans might multiply Romans numerals by converting to Arabic, multiplying and converting back

Computer usually multiply Arabic numerals by converting to binary, multiplying and converting back

Decimal number system example

The Base determines how many different symbols are needed to represent values in that base.
We use the decimal digits we learned as children to represent the first ten symbols of any base.

The order of the digits is significant.
For example, 345 does not represent the same value as 534.

base (radix)	10
digits allowed	0 1 2 3 4 5 6 7 8 9

```
9 (base 10)

9*10**0
```

```
324 (base 10)

      3 x 10**2
    + 2 x 10**1
    + 4 x 10**0
```

Humans use octal or hex to read binary number (we'll see why when we learn how to convert bases).

Octal number system

base (radix)	8
digits allowed	0 1 2 3 4 5 6 7

Examples:

345 (base 8) is really

3 x 8**2 + 4 x 8**1 + 5 x 8**0

= 192 + 32 + 5

= 229 (base 10)

1001 (base 8) is really

1 x 8**3 + 0 x 8**2 + 0 x 8**1 + 1 x 8**0

= 512 + 0 + 0 + 1

= 513 (base 10)

Hexidecimal number system

hex	decimal
0	0
1	1
.	
.	
.	
9	9
a	10
b	11
c	12
d	13
e	14
f	15

Why a-f?

Need six more symbols

Could use heart, club, diamond, spade, square and triangle

But a-f are on keyboard and it's easy to remember c is one bigger then b

Note: English has a special symbol for twelve (not ten-ee-two)like twenty-three

base (radix)	16
digits allowed	0 1 2 3 4 5 6 7 8 9 a b c d e f
example	a3 (base 16) is really a3 = a * 16^^1 + 3 * 16^^0 = a * 16 + 3 * 1 = 10 * 16 + 3 = 160 + 3 = 163 (base 10)
	3e8 (base 16) is really = 3 x 16**2 + e x 16**1 + 8 x 16**0 = 3 x 256 + 14 x 16 + 8 x 1 = 768 + 224 + 8 = 1000 (base 10)
	100 (base 16) is really = 1 x 16**2 + 0 x 16**1 + 0 x 16**0 = 1 x 256 + 0 x 16 + 0 x 1 = 256 + 0 + 0 = 256 (base 10)

- *Common MIPS syntax for hexidecimal numbers is '0x' preceding the digits `0x1234` is 1234 (Base 16).*
- *Intel uses an 'h' as a suffix of hexidecimal numerals.*

General number system (Base-B numbers)

Any number can be used as a base for a number system.

- If the number is less than (or equal) to 10, we can use a subset of the digits 0-9 for the symbols.
- if the base is greater than 10, we use letters as additional symbols so that each digit takes up only one place in the numeral.
    - Example: Hexadecimal

In each of these number systems, **the position of the symbols (digits) is important to the actual value of the numeral**. Since, we are more familiar with decimal values, we frequently wish to know the decimal value of a number that was given in a different base.

How do I convert a number from a different base to decimal?
Calculate the decimal value of each weighted symbol (digit) in the numeral and sum each of these values.

Ok, but then how do I calculate the decimal value of a single digit in the numeral?
Use the digit's position in the numeral (shown as a subscript) as the power (or the exponent) of the base and multiply that term with the digit.

Decimal value of the n^{th}-bit digit S in a Base-B numeral = $S_n * B^n$

Let's put the solution all together now for a Base-B numeral of the form.

$$S_{n-1} \quad S_{n-2} \quad . \; . \; . \quad S_2 \quad S_1 \quad S_0$$

```
        n - is the number of bits in the numeral
        B - is the base of the numeral
        S - is the symbol (digit) at that location
in the numeral
```

Decimal value of a Base-B numeral = $\mathrm{Sum}_{(i=0 \text{ to } n-1)} S_i * B^i$

Example: $abc_{\text{Base } B} = ???_{\text{Base } 10}$

GENERAL EQUATION:

$$abc_{\text{Base } B} = [\ (a * B^2) + (b * B^1) + (c * B^0)\ ]_{\text{Base } 10}$$
$$= [aB^2 + bB + c]_{\text{Base } 10}$$

decimal --> binary

Two ways:
(1) divide decimal value by 2 until the value is 0 (see book)
(2) know your powers of two and subtract

Method (2):

```
      ... 256    128  64     32  16   8   4   2   1
             d   d   d   d   d   d   d   d   d
```

E.g., 42

What is the biggest power of two that fits? 32

42-32 = 10

What fits? 8

10-8 = 2

What fits? 2

2-2 = 0 done!

```
one 32, one 8, one 2
one 32, zero 16, one 8, zero 4, one 2, zero 1
 1       0        1       0       1     0
 101010
```

Some other common base transformations

Any base --> decimal (base 10)	*Use the summation equation as given above.*
decimal --> binary (base 2)	1. Set quotient to the decimal value 2. Divide the quotient by 2 (the base) 3. Record the remainder as the least significant digit of the result. 4. Repeat steps 2 and 3 until the value of the quotient is 0 The result is the decimal equivalent of the original binary numeral.

```
example:   36Base 10
        36/2 = 18  r=0  <-- lsb
        18/2 =  9  r=0
         9/2 =  4  r=1
         4/2 =  2  r=0
         2/2 =  1  r=0
         1/2 =  0  r=1  <-- msb

         36 base 10 == 100100 base 2
                     (msb)  (lsb)

        Example: 14 base 10
        14/2 =  7  r=0  <-- lsb
         7/2 =  3  r=1
         3/2 =  1  r=1
         1/2 =  0  r=1  <-- msb
```

$14_{\text{base } 10} == 1110_{\text{base } 2}$

binary --> octal (base 8)

1. Start at the least significant symbol.
2. divide the binary digits into groups of three symbols.
3. write the octal digit for each group.

If the number of bits is not evenly divisible by 3, then add 0's at the most significant end.

```
    example:

      1 110 000 101   (binary)
      1  6   0   5    (octal)

         11 010 100   (binary)
         3   2   4    (octal)

     100 010 111   (binary)
      4   2   7    (octal)

      10 101 110   (binary)
      2   5   6    (octal)
```

binary --> hex (base 16)

It's just like binary to octal!

1. Start at the least significant symbol.
2. Divide the binary digits into groups of four symbols.
3. Write the **hex** digit for each group.

```
    example:

 1001 1110 0111 0000 (binary)
  9    e    7    0   (hexidecimal)

 1 1111 1010 0011 (binary)
 1    f    a    3   (hexidecimal)
```

<table>
<tr><td>decimal --> any base</td><td>It's just like decimal to binary using the desired base instead of 2.

1. Set quotient to the decimal value.
2. Divide the quotient by the base.
3. Record the remainder as the least significant digit of the result.
4. Repeat steps 2 and 3 until the value of the quotient is 0.</td></tr>
<tr><td>hex --> binary</td><td>Just write the 4 bit binary code for each hex digit.<pre>   example:

        3   9   c   8   (Binary)
      0011 1001 1100 1000   (Hex)</pre></td></tr>
<tr><td>octal --> binary</td><td>Just write the 3 bit binary code for each octal digit.<pre>   example:

        3   4   7   1   0   (Binary)
       011 100 111 001 000   (Hex)</pre></td></tr>
<tr><td>hex --> octal</td><td>It is easy if you do it in two steps.

1. Convert from hexidecimal to <u>binary</u>.
2. Convert the binary to octal.</td></tr>
<tr><td>decimal --> hex</td><td>Also, easy if you do it in two steps.

1. Convert from decimal to <u>binary</u>.
2. Convert the binary to hexidecimal.</td></tr>
</table>

Where the bases are powers of a common value, this transformation
is easy (like binary, base 4, octal, hexadecimal)

Examples:

```
  base 3 to base 9

     2100122 (base 3)

     One base 9 digit is substituted for each 2 base 3
digits.
     Why 2?  Answer: 3^2=9

     base
      3   9
     -------
     00   0
     01   1
     02   2
     10   3
     11   4
     12   5
     20   6
     21   7
     22   8


     2 10 01 22 (base 3)

     2  3  1  8 (base 9)
```

Explain why humans (make that computer scientists) use hex (or octal)
to read binary

```
      10110010  !=  10010010
      1011 0010     1001 0010
        b   2         9   2
```

How many 1 bits in 178?   How in b2? b has 3 + 2 has 1 = 4.

<u>On zero and negative integers</u>

In any base b, zero is

```
 ... + 0 x b**3  + 0 x b**2  + 0 x b**1  + 0 x b**0
       =   0     +    0      +     0     +    0

     = 0  (by convention only one zero)
```

Negative integers

Most humans precede number with "-" (e.g., -2000)
(Accountant, however, use parentheses: (2000)

```
     Called signed-magnitude

       e.g., -1000 into hex?

     1000  = 3 x 16**2 + e x 16**1  + 8 x 16**0

     ==> -3e8
```

Computers, well see in Chapter 4, want to do it
with just 0 and 1's, and not other symbols -,(,).

On "decimals"

What is 3.14159?

3 x 10**0 + 1x 10**-1 + 4 x 10**-2 + 1 x 10**-3 + 5 x 10**-4 + 9 x 10**-5

or

3 x 10**0 + 1/10**1 + 4/10**2 + 1/10**3 + 5/10**4 + 9/10**5

```
i=     3  2  1  0 .  -1 -2 -3 -4 -5   (place holder, value to multiply base by)
                3 .  1  4   1   5  9
```

the summation (i) of *value* x 10**i

In base ten, called "decimal"

Can do in any base the summation (i) of S x b**i

E.g. 3e.8f in hex is

3 * 16**1 + 14 * 16**0 + 8 * 16**-1 + 15*16**-2
3*16 + 14 + 8/16 + 15/256

E.g. binary 10.101

1 * 2**1 + 0 * 2**0 + 1 * 2**-1 + 0*2**-2 + 1*2**-3
2 + 0 + 1/2 + 0/4 + 1/8
2 5/8

Floating Point Representation

What range of values is needed?

very large: Avogadro's number 6.022 x 10 ** 23 atoms/mole
mass of the earth 5.98 x 10 ** 24 kilograms
speed of light 3.0 x 10 ** 8 meters/sec

very small: charge on an electron -1.60 x 10 ** (-19)

Scientific notation uses integers to represent non-integers

- used by computer systems for representation of real numbers.
- a way of representing rational numbers using integers

(used commonly to represent nonintegers in computers)

The numeral is represented by mantissa, base and exponent.

```
                             exponent
number =  mantissa x base

mantissa == fraction == significand
base == radix
```

- point is really called a radix point, for a number with a decimal base, its called a decimal point.
- all the constants given above are in scientific notation

Normalization: A rule that ensures that there is one unique form for every representable non-integer.

Normalization Rule: 1 <= mantissa < base

For binary numbers, the mantissa will always be 1.??????? x 2 ^{exponent}

In this form, the radix point is always placed one place to the right of the first significant symbol (as above).

Precision, accuracy, and significant digits

Significant digits in the mantissa tell us something about the amount of error in a measurement.
*In Scientific Notation, the number of significant digits is used to indicate the degree of **accuracy** of the measurement.*

Accuracy: a measurement (in a scientific experiment) implies a certain amount of error, depending on equipment used.

If measure the distance between two building with

```
advanced instrument:106.1345 meters +/- 0.003(3 mm)
by walking:                98 meters +/- 15 meters
```

"accuracy" is the difference between your measurement and the true, usually unknown, value.

- the +/- says you expect the true value to be within the interval
- 98 meters +/- 15 meters ==> [83, 113] meters

Significant Digits:

Tells about accuracy by approx interval (the error).

- **example**:
  a number given as 3.6 really implies that this number is in the range of 3.6 +- .05, which is 3.55 to 3.65
  This is 2 significant digits.
  3.60 really implies that this number is in the range of
  3.6 +- .005, which is 3.595 to 3.605
  This is 3 significant digits.

number of significant digits in a number
⇨ accuracy of the number is known.
The larger the number of significant digits, the better the accuracy.

Precision:

Computers (or calculators, a more familiar machine) have a **fixed precision**. No matter what accuracy of a number is known, they give lots of digits in a number. They ignore how many significant digits are involved.

- For example, if you do the operation 1.2 x 2.2. given that each number has 2 significant digits, a correct answer is

```
  1.2
x 2.2
-----
   24
+ 24
-----
  264 --> 2.64  -->   2.6   or 2.6 +- .05
```

calculator gives 2.640000000,

⇨ accuracy much higher than possible.

The result given is just the highest precision that the calculator has.

- computers only can only indicate precision and precision is not the same as accuracy.

Example:

Area of a circle is pi x radius**2
pi = 3.14159265...
radius = 8.3 meters

Area = 81.91771634183238675... WRONG!
Area = 82 meters

Recall building and sidewalk example

Computers make the problem even worse, because they can do millions of calculations, where

(1) they can do millions of calculations and you don't look at or think about each calculation

(2) the computers can print results with many digits -- but precision does not imply accuracy

(3) you believe a computer's result too much -- they don't make mistakes, right?

A whole sub-field of CS: **Numerical analysis**.

Binary Fractions

Converting binary fractions into decimal

The digits to the left of the radix point are calculated as integers (shown above) and the digits to the right are calculated as follows. The decimal values of the two halves are added together.

To convert binary (Base 2) fractions to decimal it is necessary to use negative powers of the base for each place to the right of the binary point. For example, the first digit to the right of the binary point would be multiplied by 2^{-1} or $^{1}/_{2}$, the second digit to the right of the binary point would be multiplied by 2^{-2} or $^{1}/_{4}$, and so on...

```
 f    f    . . .  f    f    f . f    f    f . . .
  n-1  n-2         2    1    0   -1   -2   -3
                             |
                             |
                              binary point
```

The decimal value is calculated in the same way as for non-fractional numbers, the exponents are now negative.

What is the decimal equivalent of 11.01011_2?

$$11.01011_2 = (1*2^1) + (1*2^0) + (0*2^{-1}) + (1*2^{-2}) + (0*2^{-3}) + (1*2^{-4}) + (1*2^{-5})$$

$$= 2 + 1 + 0 + .25 + 0 + .0625 + .03125$$

$$= 3.34375_{10}$$

example:

101.001 (binary)
1 x 2**2 + 1 x 2**0 + 1 x 2**-3
4 + 1 + 1/8
5 1/8 = 5.125 (decimal)

```
2**-1 = .5
2**-2 = .25
2**-3 = .125
2**-4 = .0625   etc.
```

example:

101.001 (octal)

1 x 8**2 + 1 x 8**0 + 1 x 8**-3
64 + 1 + 1/512
65 1/512 = 65.0019 (approx)

13.a6 (hexadecimal)

1 x 16**1 + 3 x 16**0 + a x 16**-1 + 6 x 16**-2
16 + 3 + 10/16 + 6/256
19 166/256 = 19.64 (approx)

Convert decimal fractions into binary

1. Convert the integer portion as before.
2. Convert the fraction as follows
   a. Multiply the fraction by the base (2).
   b. Record the integer of the result as the next most significant digit of the converted value.
   c. Repeat previous steps with the new fractional portion, until the result of the fractional part is zero.
3. Add the fractional portion to the integer portion.

Consider left and right of the decimal point separately.
The stuff to the left can be converted to binary as before.
Use the following algorithm to convert the fraction:

fraction	**fraction x 2**	**digit left of point**
.8	1.6	1 <-- most significant (f_{-1})
.6	1.2	1
.2	0.4	0
.4	0.8	0

.8 (it must repeat from here!)

.8 is .1100

What is the binary equivalent of 3.34375_{10}?

$3.34375_{10} = 11_2$ + the fraction part

```
fraction * base = add the integer portion
                  to the answer and continue

==================================================
.34375  *  2  = 0.68750, '0' is next digit   11.0
.6875   *  2  = 1.3750 , '1'       "         11.01
.375     *  2  = 0.750  , '0'       "        11.010
.75      *  2  = 1.50   , '1'       "       11.0101
.5       *  2  = 1.0    , '1'       "      11.01011_2
```

NON BINARY FRACTIONS

```
EXAMPLE:  give 102.3 (base 5) in base 3

*  102 (base 5) to decimal:
    1 * 5^2 + 0 * 5^1 + 2 * 5^0
      25    +    0    +   2
      27 (base 10) = 102 (base 5)

*  .3 (base 5) to decimal:
      .3 * 5^(-1)
       3/5, or .6 (decimal)
       .6 (base 10) = .3 (base 5)

*   So, 102.3 (base 5) is 27.6 (decimal)

*   27 (decimal)  to base 3
    27/3 = 9  r=0 <-- ls digit
     9/3 = 3    0
     3/3 = 1    0
     1/3 = 0    1          27 (base 10) = 1000 (base 3)

*   .6 x 3 = 1.8    1 (ms fractional digit)
    .8 x 3 = 2.4    2
    .4 x 3 = 1.2    1
    .2 x 3 = 0.6    0
    .6 x 3  this repeats the 4 digits
```

.6 (base 10) = $.\overline{1210}$

A bar over the top of the digits that repeat is a commonly used notation.

102.3 (base 5) = $1000.\overline{1210}$ (base 3)