Binary representation

There are at least 10 kinds of binary numbers; signed and unsigned

Binary numbers

- Uses only two symbols (or digits) 1 and 0
- One such digit is called a bit (from BInaryDigit)
- Using one bit, we can represent two values 0 and 1
- We need more than one bit to represent numbers larger than that

Two bits can represent 4 numbers (subscript show base):

- 0_{10} 00_{2}
- 1_{10} 01_{2}
- 2_{10} 10_{2}
- 3_{10} 11_{2}

Base 2 versus base 10

We are used to counting in base 10:

- Ten digits (0..9)
- Rightmost digit is the number of 10^0, next (going left) 10^1, then 10^2,...etc

1011_{10} is (from right to left): $1*10^0 + 1*10^1 + 0*10^2 + 1*10^3$

Base 2:

- Two digits (0,1)

1011_2 is (from right to left): $1*2^0 + 1*2^1 + 0*2^2 + 1*2^3 = 11_{10}$

Base 2 versus base 10

- In base 10, the largest digit is 9, so we need two digits to represent 10
- In base 2, the largest digit is 1, so we need two digits to represent 2_{10}
  - $10_2 = 2_{10}$
  - $(0*2^0 + 1*2^1 = 2)_{10}$
- How many bits do we need to represent a certain number?

Bits needed to represent numbers

We’ll start with unsigned numbers (positive numbers and 0).

In order to represent N numbers, we need N combinations of 1s and 0s. That means the following amount of numbers from bits:

- ($2^0 = 1$) here for completion - we don't use 0 bits to store information ;-)
- $2^1 = 2$ using **one** bit
- $2^2 = 4$ using **two** bits
- $2^3 = 8$ using **three** bits
- $2^4 = 16$ using **four** bits

How many bits do we need to represent 300_{10}

We need at least 300 combinations of 1s and 0s, so we are looking for the first power of 2 which is greater than or equal to 300.

2^9 is 512, so we need nine bits to represent the unsigned number 300.

Which number is it?

$(256 + 32 + 8 + 4 = 300)_{10}$ so the number would be:

100101100_2. For increased readability, we could write it 100 101 100.

Reading from left to right:
1*256 + **0***128 + **0***64 + **1***32 + **0***16 + **1***8 + **1***4 + **0***2 + **0***1.

And decimal numbers for 300_{10}

We need at least 300 combinations of 0-9, so we are looking for the first power of 10 which is greater than or equal to 300.

10^3 is 1000, so we need three digits to represent the unsigned number 300.

Which number is it? $(300)_{10}$

Reading from left to right:
3*100 + **0***10 + **0***1.

bits and Bytes

- 8 bits is a common unit. It is called a Byte.
- What numbers (unsigned) can we represent in a Byte?

2^8 is 256. Then there's the zero. So the numbers 0..255 (that's 256 numbers!).

This is a rule. Unsigned numbers go from 0 to 2^n-1 where n is the number of bits.

What about negative numbers?

We don’t want to sound negative, but it seems a bit thin to only be able to represent 0 and the positive numbers in a computer.

- For a number to be either positive or negative (one of two signs), we need one bit of information
- Eight bit numbers will use one bit for the sign, leaving seven bits for “values”
- Leftmost bit is usually the signbit

One way of representing also negatives

- Leftmost bit reserved for sign (0 means positive, 1 means negative)
- The negative representation of a positive number is achieved by inverting all bits (0 becomes 1 and 1 becomes 0)
- Example:
  - $0000\ 0001_2$ represents 1_{10}
  - $1111\ 1110_2$ represents -1_{10}
- What about zero? We end up with
  - $0000\ 0000_2$ represents $+0_{10}$
  - $1111\ 1111_2$ represents -0_{10}
- Possible numbers to represent are -127,... -0, +0,... +127

Another way to represent negatives

- Called two's complement - used by most modern computers
- Positive numbers are represented in "normal" binary (same as for unsigned numbers) using the bits following the signbit (which is 0 for positive numbers)
- For negative numbers, they are represented by the bits following the following operations:
  - Flip all bits of the absolute value of the number
  - Add one
- Example:
  - $1_{10} = (0000\ 0001)_2$
  - $-1_{10} = (1111\ 1110 + 1)_2 = 1111\ 1111_2$

What about zero?

- In two’s complement representation, we only have one zero
- “Trying” to represent -0 would lead to this:
    - Flip all bits of the absolute value of the number 0000 0000 gives 1111 1111
    - Add one too 1111 1111 = 0000 0000 (the carry is ignored)

```
11111 1111 (carry)
 1111 1111
+0000 0001
----------
10000 0000

The one is ignored!
```

What numbers can be represented?

- -2^{n-1} … 0 … $2^{n-1}-1$ using two's complement
- For eight bit representations, that means -128 - 127

  $-2^7 = -128$

  $2^7-1 = 127$
- 256 different numbers 128 negatives, 0 and 127 positives

Addition in unsigned and two's complement

- Works the same as in decimal (using carry)
- $(1 + 1)_2 = 10_2$ (0 but with one in carry)
- $(1 + 0)_2 = 1_2$
- $(0 + 1)_2 = 1_2$
- $(0 + 0)_2 = 0_2$

```
        11  (carry)
 0000 0011 (3₁₀)
+0000 0001 (1₁₀)
----------
 0000 0100 (4₁₀)
```

Subtraction in unsigned and two's complement

- Works the same as in decimal (using borrow)
- $(1 - 1)_2 = 0_2$ (0)
- $(1 - 0)_2 = 1_2$
- $(0 - 1)_2 = 1_2$ (borrow 10)
- $(0 - 0)_2 = 0_2$

        10 (borrow 10)
 0000 00~~1~~0 (2_{10})
-0000 0001 (1_{10})

 0000 0001 (1_{10})

Padding

- If we use eight bits, positive numbers are padded with zeros and negative numbers are padded with ones, as follows:

0000 0011 (3_{10}) (unsigned 11 means the same as 0000 0011)
1111 1111 (-1_{10}) (11 in two's complement is the same as 1111 1111)

- Extending to 16 bits, just pad:

0000 0000 0000 0011 (3_{10})
1111 1111 1111 1111 (-1_{10})

Overflow and underflow

- When programming, values have types, and types have a size (number of bits)
- In Java, for instance, `byte` has 8 bits, `short` 16 bits, `int` 32 bits and `long` 64 bits

A byte (using 2's comlement representation) can hold the values between

1000 0000 (-128_{10}) through 0111 1111 (127_{10})

Overflow and underflow

A byte (using 2's comlement representation) can hold the values between 1000 0000 (-128_{10}) through 0111 1111 (127_{10})

```
 1111 111  (carry)
 0111 1111 (127₁₀)
+0000 0001 (1₁₀)
----------
 1000 0000 (-128₁₀)
```

Overflow and underflow

A byte (using 2's comlement representation) can hold the values between 1000 0000 (-128_{10}) through 0111 1111 (127_{10})

```
 0111 1110 (borrow)
 1000 0000 (-128₁₀)
-0000 0001 (1₁₀)
----------
 0111 1111 (127₁₀)
```

Further reading

- http://www.wolframalpha.com/widgets/view.jsp?id=a291eb7ce8c6c27ed798151c4a0741bc WolframAlpha - calculator
- https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html Oracle tutorial - Java datatypes
- https://en.wikipedia.org/wiki/Two%27s_complement Wikipedia - Two's compliment
- https://en.wikipedia.org/wiki/Binary_number Wikipedia - Binary number
- https://www.ugrad.cs.ubc.ca/~cs121/2009W1/Handouts/signed-binary-decimal-conversions.html Signed Binary/Decimal Conversion Using the Two's Complement Representation (University of British Columbia, Canada)
- http://www.exploringbinary.com/twos-complement-converter/ - Exploring Binary - Decimal/Two’s Complement Converter
- https://en.wikipedia.org/wiki/Ones%27_complement One’s complement