Positional number representation

Example: 1823 in base 10 (alphabet: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9)

\[ 1823_{10} = 3 \times 10^3 + 2 \times 10^2 + 8 \times 10^1 + 1 \times 10^0 \]

\[ \begin{array}{cccccc}
\text{position} & 4 & 3 & 2 & 1 & 0 \\
\text{number} & 1 & 8 & 2 & 3 & \text{multiplier} = 10^3 \\
\end{array} \]

Example: 1823 in base 13 (alphabet: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C)

\[ 1823_{13} = 3 \times 13^3 + 2 \times 13^2 + 8 \times 13^1 + 1 \times 13^0 \]

\[ \begin{array}{cccccc}
\text{position} & 4 & 3 & 2 & 1 & 0 \\
\text{number} & A & A & 3 & \text{multiplier} = 13^3 \\
\end{array} \]

The above examples also show how to convert a number in a generic base \( b \) into a number in base 10.

---

CMPE 110 — Winter 2004

Integer Number Representation

Integer number representations

- positional number representation and base conversion
- unsigned and signed binary numbers
- binary addition and subtraction

---

Positional number representation

Positional number representation description:

- a specific number is selected as the *base* (e.g., 10)
- an *alphabet* of *base-1* symbols plus the 0 must be defined to represent all numbers < *base* (e.g., 0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
- numbers ≥ *base* are represented by an ordered sequence of 2 or more symbols (digits) from the selected alphabet
- by *ordered* sequence we mean that each symbol appears in a specific *position* in the representation, and that *positions are numbered* starting from 0 in the rightmost position
- a represented number is the sum of all digits, each multiplied by *base* raised to the power of their *position*

---

Positional number representation

Different ways of representing the same number:

- \( 382 = \text{MCXXXIII}_{\text{Roman}} = 71F_{16} = 1110001111_2 = 343_8 = 11111111_1 \) (1 repeated 1823 times)

Among these, Roman is the only *non-positional* representation.
Binary numbers

**Sign and magnitude:** One bit (usually the leftmost) is used to indicate the sign, for instance 0 meaning “+” and 1 meaning “-”.

Example: \(123_{10} = 01111011_2, -123_{10} = 11111011_2\)

Shortcomings:
- In arithmetic operations, the computer first has to look at the signs of both numbers to decide whether to add them or subtract one from the other, and what sign the result will bear.
- There is a positive and a negative zero:
  \(+0 = 00000000, -0 = 10000000\)

Positional number representation

Conversion of a number \(N\) from base 10 to a generic base \(b\):

\[N = q \cdot b^i + r\]

\(q = \lfloor N/b \rfloor\)
\(r = \text{ith digit of } N_b\)

Example: Converting \(124_{10}\) to base \(b = 3\)

<table>
<thead>
<tr>
<th>(i)</th>
<th>(N)</th>
<th>(q)</th>
<th>(r)</th>
<th>(N_b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1111</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1110</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1101</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>1000</td>
</tr>
</tbody>
</table>

The number of digits required to represent a number \(N_{10}\) in base \(b\) is \(\lceil \log_b N \rceil\).

In the example, \(\log_3 124 = 4.387 \ldots\) and \(\lceil 4.387 \ldots \rceil = 5\).

Binary numbers

**One's complement:** Given a binary number on \(n\) bits, its one's complement is obtained by independently subtracting each digit from one (which is equivalent to changing all 1's into 0's and vice versa).

Example: \(312_{10} = 01111011_2\)

\[
\begin{array}{c}
1111111 \\
01111011 \\
\hline
10000000
\end{array}
\]

Given a positive binary number \(N\), we can use its one's complement to represent its negative \(-N\).

Shortcomings:
- There is a positive and a negative zero:
  \(+0 = 00000000, -0 = 11111111\)
- Arithmetic operations are still complicated \((a + (\neg a)) \neq 0\)

**Binary numbers - base = 2**

- Why do computers use binary numbers?
- *Unsigned* binary numbers - only one choice

Example: \(8_{10} = 1000_2\)

The largest number that can be represented with \(n\) bits is \(2^n - 1\).

- *Signed* binary numbers - three main choices:
  - "Sign and magnitude"
  - "One's complement"
  - "Two's complement"
Binary numbers

Sign bit and sign extension

- in two's complement, the most significant bit is always 0 in positive numbers and always 1 in negative numbers - hence it is called the "sign bit"
- positive numbers have an infinite number of 0's to the left of their msb, and negative numbers have an infinite number of 1's to the left of their msb,
- when moving a two's complement number from an n-bit representation to an m-bit representation with m > n, we must remember to sign extend it.

Example: loading the signed 8-bit number -86\textsubscript{10} into a 16-bit register,

-86\textsubscript{10} on 8 bits: 1010 1010
-86\textsubscript{10} on 16 bits: 1111 1111 1010 1010

Binary numbers

two's complement: given a binary number on n bits, its two's complement is obtained by subtracting the number from \( 2^n \). This is equivalent to adding one to its one's complement.

Example: \( 123_{10} = 01111011_2 \)

<table>
<thead>
<tr>
<th>original number</th>
<th>01111011</th>
</tr>
</thead>
<tbody>
<tr>
<td>its one's complement</td>
<td>+</td>
</tr>
<tr>
<td>add one</td>
<td>=</td>
</tr>
<tr>
<td>two's complement</td>
<td></td>
</tr>
</tbody>
</table>

Given a positive binary number \( N \), we can use its two's complement to represent its negative \(-N\).

Shortcut to convert a binary number into its two's complement (different from the book): starting from the right, keep all the zeros and the first one, then invert all remaining bits.

Binary numbers

Addition and subtraction in two's complement

addition: bit-by-bit from right to left (carry)

subtraction: add the first operand to the two's complement of the second operand.

This is based on the fact that: \( a - b \) can be done as \( a + (-b) \), where \(-b\) is \( b\)'s two's complement.

Works regardless of the signs and relative magnitudes of the operands.

Binary numbers

More on two's complement numbers

Shortcomings:
- the range is asymmetric:
  - for an n-bit number goes from \(-2^{n-1}\) to \(+2^{n-1} - 1\)
- the logical negation (bitwise) is different from the arithmetic negation

Advantages:
- there is only one zero
- arithmetic operations are simple
Hexadecimal and octal numbers

Hexadecimal numbers: base = 16
- alphabet: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
- each hex digit represents a group of four bits, Example: converting the 24-bit binary number 101100000110011010011 to hex,
  bin: 1011 0000 0110 0101 1101 0011
  hex: 

Octal numbers: base = 8
- alphabet: 0, 1, 2, 3, 4, 5, 6, 7
- each hex digit represents a group of three bits, Example: converting the 24-bit binary number 101100000110011010011 to octal,
  bin: 101 100 000 110 010 111 010 011
  octal: 

Recommended exercises
- Ex. 4, 4, 2, 4, 4, 4, 4, 5, 6, 7, 8, 9, 10, 11, 16
- Exercise T01.A:
  Convert the number 12310 into base b for all b = 2, 3, 5, 8, 11, 16
- Exercise T01.B:
  Convert the number 12310 into base 10 for all b = 4, 7, 8, 10, 12, 16

Overflow
- with an infinite-digit representation overflow has no meaning
- with a n-bit representation, overflow occurs when the result of an operation is less than \( -2^{n-1} \) or greater than \( 2^{n-1} - 1 \)
- in addition overflow can occur when the two operands have the same sign
- in subtraction overflow can occur when the two operands have different signs

<table>
<thead>
<tr>
<th>Operation</th>
<th>A</th>
<th>B</th>
<th>OF if result is</th>
</tr>
</thead>
<tbody>
<tr>
<td>A+B</td>
<td>≥ 0</td>
<td>≥ 0</td>
<td>≥ 0</td>
</tr>
<tr>
<td>A+B</td>
<td>&lt; 0</td>
<td>&lt; 0</td>
<td>&lt; 0</td>
</tr>
<tr>
<td>A*B</td>
<td>≥ 0</td>
<td>&lt; 0</td>
<td>≥ 0</td>
</tr>
<tr>
<td>A-B</td>
<td>&lt; 0</td>
<td>≥ 0</td>
<td>≥ 0</td>
</tr>
</tbody>
</table>

Binary numbers

Example: adding 22010 + 11010 represented as 8 bit unsigned binary numbers

\[
\begin{align*}
\text{carry} & \\
220_{10} &+ 1101100_{10} &+ \\
110_{10} &01101110 &- \\
\hline
330_{10} & & \\
\end{align*}
\]

Example: adding 10010 + 5010 represented as 8 bit two's complement binary numbers

\[
\begin{align*}
\text{carry} & \\
100_{10} &+ 01100100 &+ \\
50_{10} &00110010 &- \\
\hline
150_{10} & & \\
\end{align*}
\]