Представление чисел в памяти

Иван Казменко

СПбГУ, 2019–2020 учебный год, первый курс

Вторник, 24 сентября 2019 года

Оглавление

1. Представление целых чисел
    - Системы счисления
    - Целочисленные типы данных
    - Отрицательные числа
    - Битовые операции
    - Работа с отдельными битами
2. Представление вещественных чисел
    - Как хранить вещественное число?
    - Вещественные типы данных
    - Числа с плавающей точкой
    - Специальные значения
    - Примеры

Оглавление

1. Представление целых чисел
   - Системы счисления
   - Целочисленные типы данных
   - Отрицательные числа
   - Битовые операции
   - Работа с отдельными битами
2. Представление вещественных чисел
   - Как хранить вещественное число?
   - Вещественные типы данных
   - Числа с плавающей точкой
   - Специальные значения
   - Примеры

Системы счисления

Постановка задачи:
требуется записывать числа и производить операции с ними.

Критерии качества:

- Наглядность.
- Краткость.
- Удобство арифметических действий.

Системы счисления

Постановка задачи:
требуется записывать числа и производить операции с ними.

Критерии качества:

- Наглядность.
- Краткость.
- Удобство арифметических действий.

Системы счисления

Постановка задачи:
требуется записывать числа и производить операции с ними.

Критерии качества:

- Наглядность: для маленьких чисел.
- Краткость: для очень маленьких чисел.
- Удобство арифметических действий: для маленьких чисел.

Пример 1 — унарная система счисления.
Число записывается палочками, стоящими подряд: $|$, $||$, $|||$, $||||$, $\ldots$

Системы счисления

Постановка задачи:
требуется записывать числа и производить операции с ними.

Критерии качества:

- Наглядность: до нескольких тысяч, нужна привычка.
- Краткость: до нескольких тысяч.
- Удобство арифметических действий: сложные правила.

Пример 2 — римские числа.
Вводятся специальные символы для обозначения больших количеств: $5 = V$, $10 = X$, $50 = L$, $100 = C$, $500 = D$, $1000 = M$.
Позиции символов также имеют значение: например,
$VI = V + I = 6$, но $IV = V - I = 4$.

Системы счисления

Постановка задачи:
требуется записывать числа и производить операции с ними.

Критерии качества:

- Наглядность: для не очень больших чисел, нужна привычка.
- Краткость: для не очень больших чисел.
- Удобство арифметических действий: легко складывать и вычитать, с таблицей умножения можно производить умножение и деление в столбик.

Пример 3 — позиционные системы счисления: десятичная.
Значение имеют и символ (цифра), и позиция (степень основания): $153_{10} = 3 \cdot 1 + 5 \cdot 10 + 1 \cdot 100$.

Системы счисления

Постановка задачи:
требуется записывать числа и производить операции с ними.

Критерии качества:

- Наглядность: для не очень больших чисел, нужна привычка.
- Краткость: для не очень больших чисел.
- Удобство арифметических действий: легко складывать и вычитать, таблица умножения простая, а значит, действия легко реализуются на уровне микросхемы.

Пример 4 — позиционные системы счисления: двоичная.
Значение имеют и символ (цифра), и позиция (степень основания): $100101_2 = 1 + 4 + 32 = 37$.

Целочисленные типы данных

Рассмотрим числа, в двоичной записи которых не больше k цифр. Отведём место ровно под k цифр (битов).

Пример — типы языка GNU C++ для архитектуры x86.

Целочисленные типы данных

Рассмотрим числа, в двоичной записи которых не больше k цифр. Отведём место ровно под k цифр (битов).

Пример — типы языка GNU C++ для архитектуры x86.

Тип	**Байты**	**Биты**	**Минимум**	**Максимум**
`unsigned char`	1	8	0	255
`unsigned short`	2	16	0	65 535
`unsigned long`	4	32	0	$2^{32} - 1$
`unsigned long long`	8	64	0	$2^{64} - 1$

Целочисленные типы данных

Рассмотрим числа, в двоичной записи которых не больше k цифр. Отведём место ровно под k цифр (битов).

Пример — типы языка GNU C++ для архитектуры x86.

Тип	**Байты**	**Биты**	**Минимум**	**Максимум**
`unsigned char`	1	8	0	255
`unsigned short`	2	16	0	65 535
`unsigned long`	4	32	0	$2^{32}-1$
`unsigned long long`	8	64	0	$2^{64}-1$

Адресовать память обычно можно с точностью до байта.
В каком порядке идут байты в целом числе?

- Big-endian: $1000 = 11\,1110\,1000_2$, код $[0000\,0011]$ $[1110\,1000]$.
  Удобно для человеческого восприятия.
- Little-endian: $1000 = 11\,1110\,1000_2$, код $[1110\,1000]$ $[0000\,0011]$.
  Если число помещается в меньший тип, адрес остаётся тем же.

Целочисленные типы данных

Рассмотрим числа, в двоичной записи которых не больше k цифр. Отведём место ровно под k цифр (битов).

Пример — типы языка GNU C++ для архитектуры x86.

Тип	**Байты**	**Биты**	**Минимум**	**Максимум**
`unsigned char`	1	8	0	255
`unsigned short`	2	16	0	65 535
`unsigned long`	4	32	0	$2^{32}-1$
`unsigned long long`	8	64	0	$2^{64}-1$
`signed char`	1	8	-128	127
`signed short`	2	16	$-32\,768$	32 767
`signed long`	4	32	-2^{31}	$2^{31}-1$
`signed long long`	8	64	-2^{63}	$2^{63}-1$

Для отрицательных чисел уменьшим диапазон на один бит. Этот бит мы используем для хранения знака.

Отрицательные числа

Если просто хранить знак, то:

- У нуля будет два различных представления ($+0$ и -0).
- При сложении и вычитании придётся разбирать случаи.

Отрицательные числа

Двоичный дополнительный код — менее нагляден, но проще для механических вычислений.

Код	**Без знака**	**Со знаком**
0000 0000	0	0
0000 0001	1	+1
0000 0010	2	+2
...	...	...
0111 1110	126	+126
0111 1111	127	+127
1000 0000	128	−128
1000 0001	129	−127
1000 0010	130	−126
...	...	...
1111 1110	254	−2
1111 1111	255	−1

Отрицательные числа

Двоичный дополнительный код — менее нагляден, но проще для механических вычислений.

Код	Без знака	Со знаком	Разность
0000 0000	0	0	0
0000 0001	1	+1	0
0000 0010	2	+2	0
...	...	...	...
0111 1110	126	+126	0
0111 1111	127	+127	0
1000 0000	128	−128	256
1000 0001	129	−127	256
1000 0010	130	−126	256
...	...	...	...
1111 1110	254	−2	256
1111 1111	255	−1	256

Отрицательные числа

Двоичный дополнительный код — менее нагляден, но проще для механических вычислений.

1. Интерпретация: арифметика по модулю 2^k.
   Хранится остаток числа по модулю 2^k. Можно интерпретировать его или как число без знака, или как число со знаком.

Пример: $1001\,0101_2 = 1 + 4 + 16 + 128 = 149$.

- Число без знака: какое число от 0 до 255 имеет остаток 149 от деления на $2^8 = 256$? Это число 149.
- Число со знаком: какое число от -128 до $+127$ имеет остаток 149 от деления на $2^8 = 256$? Это число -107.

Отрицательные числа

Двоичный дополнительный код — менее нагляден, но проще для механических вычислений.

2. Интерпретация: вычитание 2^k.
   Если число получается больше максимально возможного (то есть $2^{k-1} - 1$), отнимем от него 2^k.

Пример: $1001\,0101_2 = 1 + 4 + 16 + 128 = 149$.

$149 > 127$, значит, значение равно $149 - 256 = -107$.

Отрицательные числа

Двоичный дополнительный код — менее нагляден, но проще для механических вычислений.

3. Интерпретация: отрицательное основание для старшего бита.
   Обычно бит с номером i домножается на 2^i. А теперь старший бит (с номером $k-1$) домножается не на 2^{k-1}, а на -2^{k-1}.

Пример: $1001\,0101_2 = 1 + 4 + 16 + 128 = 149$.

Код $1001\,0101$ даёт $1 + 4 + 16 - 128 = -107$.

Отрицательные числа

Двоичный дополнительный код — менее нагляден, но проще для механических вычислений.

4. Интерпретация: механический способ.
   Чтобы получить из кода отрицательного числа его модуль, инвертируем все биты, а затем прибавим единицу.

Пример: $1001\,0101_2 = 1 + 4 + 16 + 128 = 149$.

$$\begin{aligned} \text{Инверсия } 1001\,0101_2 + 1 &= \\ 0110\,1010_2 + 1 &= \\ 0110\,1011_2 \quad &= \\ 1 + 2 + 8 + 32 + 64 \quad &= \\ 107 \end{aligned}$$

Отрицательные числа

Двоичный дополнительный код — менее нагляден, но проще для механических вычислений.

Сложение и умножение делаются в столбик.

$$
\begin{array}{rrl}
 & 1001\,0101_2 & (= -107) \\
+ & 0110\,1111_2 & (= +111) \\
= & 1\,0000\,0100_2 & (\text{оставим последние } k \text{ цифр результата}) \\
\to & 0000\,0100_2 & (= +4)
\end{array}
$$

Отрицательные числа

Двоичный дополнительный код — менее нагляден, но проще для механических вычислений.

Сложение и умножение делаются в столбик.

$$
\begin{array}{rrl}
 & 1111\,1011_2 & (= -5) \\
\times & 0000\,0111_2 & (= +7) \\
= & 1111\,1011_2 & \\
+ & 1\,1111\,0110_2 & \\
+ & 11\,1110\,1100_2 & \\
= & 110\,1101\,1101_2 & (\text{оставим последние } k \text{ цифр результата}) \\
\to & 1101\,1101_2 & (= -35)
\end{array}
$$

Битовые операции

Представление целых чисел даёт возможность легко выполнять некоторые действия.

1. Инвертирование всех битов.
2. Побитовая операция AND.
3. Побитовая операция OR.
4. Побитовая операция XOR.
5. Побитовый сдвиг влево.
6. Логический сдвиг вправо.
7. Арифметический сдвиг вправо.

Битовые операции

Представление целых чисел даёт возможность легко выполнять некоторые действия.

1. Инвертирование всех битов.

$$\texttt{a} = 1001\,0011,$$
$$\texttt{\~{}a} = 0110\,1100.$$

A	~A
0	1
1	0

Битовые операции

Представление целых чисел даёт возможность легко выполнять некоторые действия.

2. Побитовая операция AND.

$$\begin{aligned} \texttt{a} &= 1001\,0011, \\ \texttt{b} &= 1100\,1010, \\ \texttt{a \& b} &= 1000\,0010. \end{aligned}$$

A	B	A & B
0	0	0
0	1	0
1	0	0
1	1	1

Битовые операции

Представление целых чисел даёт возможность легко выполнять некоторые действия.

3. Побитовая операция OR.

$$\begin{aligned} \texttt{a} &= 1001\,0011, \\ \texttt{b} &= 1100\,1010, \\ \texttt{a | b} &= 1101\,1011. \end{aligned}$$

A	B	A | B
0	0	0
0	1	1
1	0	1
1	1	1

Битовые операции

Представление целых чисел даёт возможность легко выполнять некоторые действия.

4. Побитовая операция XOR.

$$\begin{aligned} \texttt{a} &= 1001\,0011, \\ \texttt{b} &= 1100\,1010, \\ \texttt{a ^ b} &= 0101\,1001. \end{aligned}$$

A	B	A ^ B
0	0	0
0	1	1
1	0	1
1	1	0

Битовые операции

Представление целых чисел даёт возможность легко выполнять некоторые действия.

5. Побитовый сдвиг влево.

$$\begin{aligned} \texttt{a} &= 1001\,0011, \\ \texttt{b} &= 3, \\ \texttt{a << b} &= 1001\,1000. \end{aligned}$$

Все биты сдвигаются влево `b` раз.
Старший бит пропадает.
Младший бит становится нулём.

Битовые операции

Представление целых чисел даёт возможность легко выполнять некоторые действия.

6. Логический сдвиг вправо.

$$
\begin{aligned}
\mathtt{a} &= 1001\,0011,\\
\mathtt{b} &= 2,\\
\mathtt{a}\ \ \mathtt{>>>}\ \ \mathtt{b} &= 0010\,0100.
\end{aligned}
$$

Все биты сдвигаются вправо $\mathtt{b}$ раз.
Младший бит пропадает.
Старший бит становится нулём.

Битовые операции

Представление целых чисел даёт возможность легко выполнять некоторые действия.

7. Арифметический сдвиг вправо.

$$\begin{aligned} \texttt{a} &= 1001\,0011, \\ \texttt{b} &= 2, \\ \texttt{a >> b} &= 1110\,0100. \end{aligned}$$

Все биты сдвигаются вправо b раз.
Младший бит пропадает.
Старший (знаковый) бит не меняется.

Работа с отдельными битами

Биты нумеруются с нуля, начиная с младшего.
Номер бита равен степени двойки, на которую домножается этот бит.

$$\text{номера битов:} \quad 7654\,3210$$
$$\mathsf{a} = 1001\,0011$$

Операции с отдельными битами:

1. Как получить значение бита с номером k:
`get (a, k): (a >> k) & 1`
2. Как установить бит с номером k в единицу:
`set1 (a, k): a ← a | (1 << k)`
3. Как установить бит с номером k в ноль:
`set0 (a, k): a ← a & ~(1 << k)`
4. Как установить бит с номером k в значение v:
`set (a, k, v): a ← (a & ~(1 << k)) | (v << k)`

Работа с отдельными битами

Биты нумеруются с нуля, начиная с младшего.
Номер бита равен степени двойки, на которую домножается этот бит.

номера битов: $7654\,3210$

$\mathtt{a} = 1001\,0011$

$k = 3$

$\mathtt{a} \;\mathtt{>>}\; \mathtt{k} = 0001\,0010$

значение $= 0000\,0000$

Операции с отдельными битами:

1. Как получить значение бита с номером `k`:
   `get (a, k): (a >> k) & 1`
2. Как установить бит с номером `k` в единицу:
   `set1 (a, k): a ← a | (1 << k)`
3. Как установить бит с номером `k` в ноль:
   `set0 (a, k): a ← a & ~(1 << k)`
4. Как установить бит с номером `k` в значение `v`:
   `set (a, k, v): a ← (a & ~(1 << k)) | (v << k)`

Работа с отдельными битами

Биты нумеруются с нуля, начиная с младшего.
Номер бита равен степени двойки, на которую домножается этот бит.

номера битов: 7654 3210

`a` = 1001 0011

$k = 3$

`1 << k` = 0000 1000

`a` ← 1001 1011

Операции с отдельными битами:

1. Как получить значение бита с номером `k`:
   `get (a, k): (a >> k) & 1`
2. Как установить бит с номером `k` в единицу:
   `set1 (a, k): a ← a | (1 << k)`
3. Как установить бит с номером `k` в ноль:
   `set0 (a, k): a ← a & ~(1 << k)`
4. Как установить бит с номером `k` в значение `v`:
   `set (a, k, v): a ← (a & ~(1 << k)) | (v << k)`

Работа с отдельными битами

Биты нумеруются с нуля, начиная с младшего.
Номер бита равен степени двойки, на которую домножается этот бит.

номера битов: $7654\,3210$

`a` $= 1001\,0011$

$k = 3$

`~(1 << k)` $= 1111\,0111$

`a` $\leftarrow 1001\,0011$

Операции с отдельными битами:

1. Как получить значение бита с номером `k`:
   `get (a, k): (a >> k) & 1`
2. Как установить бит с номером `k` в единицу:
   `set1 (a, k): a` $\leftarrow$ `a | (1 << k)`
3. Как установить бит с номером `k` в ноль:
   `set0 (a, k): a` $\leftarrow$ `a & ~(1 << k)`
4. Как установить бит с номером `k` в значение `v`:
   `set (a, k, v): a` $\leftarrow$ `(a & ~(1 << k)) | (v << k)`

Работа с отдельными битами

Биты нумеруются с нуля, начиная с младшего.
Номер бита равен степени двойки, на которую домножается этот бит.

номера битов: $7654\,3210$ $\quad$ `v << k` $= 0000\,\mathrm{v}000$

`a` $= 1001\,0011$ $\quad k = 3 \quad$ `a` $\leftarrow 1001\,\mathrm{v}011$

Операции с отдельными битами:

1. Как получить значение бита с номером `k`:
   `get (a, k): (a >> k) & 1`
2. Как установить бит с номером `k` в единицу:
   `set1 (a, k): a ← a | (1 << k)`
3. Как установить бит с номером `k` в ноль:
   `set0 (a, k): a ← a & ~(1 << k)`
4. Как установить бит с номером `k` в значение `v`:
   `set (a, k, v): a ← (a & ~(1 << k)) | (v << k)`

Работа с отдельными битами

Биты нумеруются с нуля, начиная с младшего.
Номер бита равен степени двойки, на которую домножается этот бит.

номера битов: $\quad 7654\,3210$

$$\mathtt{a} = 1001\,0011$$

Операции с отдельными битами:

1. Как получить значение бита с номером `k`:
   `get (a, k): (a >> k) & 1`
2. Как установить бит с номером `k` в единицу:
   `set1 (a, k): a ← a | (1 << k)`
3. Как установить бит с номером `k` в ноль:
   `set0 (a, k): a ← a & ~(1 << k)`
4. Как установить бит с номером `k` в значение `v`:
   `set (a, k, v): a ← (a & ~(1 << k)) | (v << k)`

Оглавление

1. Представление целых чисел
   - Системы счисления
   - Целочисленные типы данных
   - Отрицательные числа
   - Битовые операции
   - Работа с отдельными битами
2. Представление вещественных чисел
   - Как хранить вещественное число?
   - Вещественные типы данных
   - Числа с плавающей точкой
   - Специальные значения
   - Примеры

Как хранить вещественное число?

Постановка задачи: требуется хранить вещественные числа и производить с ними вычисления.

Критерии качества:

- Диапазон значений.
- Удобство хранения.
- Удобство арифметических действий.

Как хранить вещественное число?

Постановка задачи: требуется хранить вещественные числа и производить с ними вычисления.

Критерии качества:

- Диапазон значений.
- Удобство хранения.
- Удобство арифметических действий.

Как хранить вещественное число?

Постановка задачи: требуется хранить вещественные числа и производить с ними вычисления.

Критерии качества:

- Диапазон значений: рациональные числа с небольшими числителем и знаменателем.
- Удобство хранения: да.
- Удобство арифметических действий: приведение к общему знаменателю: числитель и знаменатель быстро растут.

Способ 1 — рациональные числа.
Число представляется в виде $\frac{P}{Q}$, где P и Q — целые: $\frac{0}{1}$, $\frac{5}{3}$, $\frac{-6}{11}$, ...
Где применяется: точное решение некоторых математических задач.

Как хранить вещественное число?

Постановка задачи: требуется хранить вещественные числа и производить с ними вычисления.

Критерии качества:

- Диапазон значений: почти любое число хранится приближённо.
- Удобство хранения: да.
- Удобство арифметических действий: как с целыми.

Способ 2 — числа с фиксированной точкой.
Отдельно хранятся целая часть и дробная часть.
Например, $X = A.B = A + \frac{1}{65\,536} B$, где $0 \leqslant A, B < 65\,536$.
Можно считать, что хранятся целые числа, но их значения при использовании следует делить на $65\,536$.
Где применяется: METAPOST, библиотека Allegro, Java BigDecimal.

Как хранить вещественное число?

Постановка задачи: требуется хранить вещественные числа и производить с ними вычисления.

Критерии качества:

- Диапазон значений: почти любое число хранится приближённо, можно хранить очень большие и очень маленькие по модулю числа.
- Удобство хранения: да.
- Удобство арифметических действий: почти как с целыми.

Способ 3 — числа с плавающей точкой.
Хранятся первые несколько двоичных знаков числа, а также степень двойки (небольшое положительное или отрицательное число), на которую нужно домножить.
Например, $X = 1.00101_2 \cdot 2^{100}$, $Y = -1 = -1.00000_2 \cdot 2^0$.
Та же конструкция с десятичными числами: $X = 1.23456 \cdot 10^{100}$, $Y = -1 = -1.00000 \cdot 10^0$.

Вещественные типы данных

Отведём место ровно под k первых двоичных цифр, ещё p битов на степень двойки и ещё один бит на знак числа.

Пример — типы языка GNU C++ для архитектуры x86.

Вещественные типы данных

Отведём место ровно под k первых двоичных цифр, ещё p битов на степень двойки и ещё один бит на знак числа.

Пример — типы языка GNU C++ для архитектуры x86.

Тип	**Байты**	**s**	**p**	**k**	**Минимум**	**Максимум**
`float`	4	1	8	23	$\pm 1.5 \cdot 10^{-45}$	$\pm 3.4 \cdot 10^{+38}$
`double`	8	1	11	52	$\pm 5.0 \cdot 10^{-324}$	$\pm 1.7 \cdot 10^{+308}$
`long double`	10	1	15	64	$\pm 3.4 \cdot 10^{-4932}$	$\pm 1.1 \cdot 10^{+4932}$

Числа с плавающей точкой

Представление числа:

$$X = (-1)^S \times 1.M \times 2^{E - E_0}$$

Значение $E_0 = 2^{p-1} - 1$.
Мы будем рассматривать примеры для типа `float` ($E_0 = 127$).

Числа с плавающей точкой

Представление числа:

$$X = (-1)^S \times 1.M \times 2^{E-E_0}$$

Значение $E_0 = 2^{p-1} - 1$.
Мы будем рассматривать примеры для типа `float` ($E_0 = 127$).

Пример 1: число
$1 = (-1)^0 \times 1.0000\ldots00 \times 2^{127-127}$.

```
seee eeee emmm mmmm mmmm mmmm mmmm mmmm
|[-- ---- ][-- ---- ---- ---- ---- ----]
0011 1111 1000 0000 0000 0000 0000 0000
```

Числа с плавающей точкой

Представление числа:

$$X = (-1)^S \times 1.M \times 2^{E-E_0}$$

Значение $E_0 = 2^{p-1} - 1$.
Мы будем рассматривать примеры для типа `float` ($E_0 = 127$).

Пример 1: число
$1 = (-1)^0 \times 1.0000\ldots00 \times 2^{127-127}$.

```
seee eeee emmm mmmm mmmm mmmm mmmm mmmm
|[-- ---- ][-- ---- ---- ---- ---- ----]
0011 1111 1000 0000 0000 0000 0000 0000
```

Числа с плавающей точкой

Представление числа:

$$X = (-1)^S \times 1.M \times 2^{E-E_0}$$

Значение $E_0 = 2^{p-1} - 1$.
Мы будем рассматривать примеры для типа `float` ($E_0 = 127$).

Пример 1: число
$1 = (-1)^0 \times 1.0000\ldots00 \times 2^{127-127}$.

```
seee eeee emmm mmmm mmmm mmmm mmmm mmmm
|[-- ---- ][-- ---- ---- ---- ---- ----]
0011 1111 1000 0000 0000 0000 0000 0000
```

Числа с плавающей точкой

Представление числа:

$$X = (-1)^S \times 1.M \times 2^{E-E_0}$$

Значение $E_0 = 2^{p-1} - 1$.
Мы будем рассматривать примеры для типа `float` ($E_0 = 127$).

Пример 1: число
$1 = (-1)^0 \times 1.0000\ldots00 \times 2^{127-127}$.

```
seee eeee emmm mmmm mmmm mmmm mmmm mmmm
|[-- ---- ][-- ---- ---- ---- ---- ----]
0011 1111 1000 0000 0000 0000 0000 0000
```

Числа с плавающей точкой

Представление числа:

$$X = (-1)^S \times 1.M \times 2^{E-E_0}$$

Значение $E_0 = 2^{p-1} - 1$.
Мы будем рассматривать примеры для типа `float` ($E_0 = 127$).

Пример 2: число
$-5.5 = (-1)^1 \times 1.0110\ldots00 \times 2^{129-127}$.

```
seee eeee emmm mmmm mmmm mmmm mmmm mmmm
|[-- ---- ][-- ---- ---- ---- ---- ----]
1100 0000 1011 0000 0000 0000 0000 0000
```

Числа с плавающей точкой

Представление числа:

$$X = (-1)^S \times 1.M \times 2^{E-E_0}$$

Значение $E_0 = 2^{p-1} - 1$.
Мы будем рассматривать примеры для типа `float` ($E_0 = 127$).

Пример 2: число
$-5.5 = (-1)^1 \times 1.0110\ldots00 \times 2^{129-127}$.

```
seee eeee emmm mmmm mmmm mmmm mmmm mmmm
|[-- ---- ][-- ---- ---- ---- ---- ----]
1100 0000 1011 0000 0000 0000 0000 0000
```

Числа с плавающей точкой

Представление числа:

$$X = (-1)^S \times 1.M \times 2^{E-E_0}$$

Значение $E_0 = 2^{p-1} - 1$.
Мы будем рассматривать примеры для типа `float` ($E_0 = 127$).

Пример 2: число
$-5.5 = (-1)^1 \times 1.0110\ldots00 \times 2^{129-127}$.

```
seee eeee emmm mmmm mmmm mmmm mmmm mmmm
|[-- ---- ][-- ---- ---- ---- ---- ----]
1100 0000 1011 0000 0000 0000 0000 0000
```

Числа с плавающей точкой

Представление числа:

$$X = (-1)^S \times 1.M \times 2^{E-E_0}$$

Значение $E_0 = 2^{p-1} - 1$.
Мы будем рассматривать примеры для типа `float` ($E_0 = 127$).

Пример 2: число
$-5.5 = (-1)^1 \times 1.0110\ldots00 \times 2^{129-127}$.

```
seee eeee emmm mmmm mmmm mmmm mmmm mmmm
|[-- ---- ][-- ---- ---- ---- ---- ----]
1100 0000 1011 0000 0000 0000 0000 0000
```

Числа с плавающей точкой

Представление числа:

$$X = (-1)^S \times 1.M \times 2^{E-E_0}$$

Значение $E_0 = 2^{p-1} - 1$.
Мы будем рассматривать примеры для типа `float` ($E_0 = 127$).

Пример 3: число
$\frac{1}{5} = (-1)^0 \times 1.10011001100110011001100110011\ldots \times 2^{124-127}$.

```
seee eeee emmm mmmm mmmm mmmm mmmm mmmm
|[-- ---- ][-- ---- ---- ---- ---- ----]
0011 1110 0100 1100 1100 1100 1100 1101
```

Числа с плавающей точкой

Представление числа:

$$X = (-1)^S \times 1.M \times 2^{E-E_0}$$

Значение $E_0 = 2^{p-1} - 1$.
Мы будем рассматривать примеры для типа `float` ($E_0 = 127$).

Пример 3: число
$\frac{1}{5} = (-1)^0 \times 1.10011001100110011001100110011\ldots \times 2^{124-127}$.

```
seee eeee emmm mmmm mmmm mmmm mmmm mmmm
|[-- ---- ][-- ---- ---- ---- ---- ----]
0011 1110 0100 1100 1100 1100 1100 1101
```

Числа с плавающей точкой

Представление числа:

$$X = (-1)^S \times 1.M \times 2^{E-E_0}$$

Значение $E_0 = 2^{p-1} - 1$.
Мы будем рассматривать примеры для типа `float` ($E_0 = 127$).

Пример 3: число
$\frac{1}{5} = (-1)^0 \times 1.10011001100110011001100110011\ldots \times 2^{124-127}$.

```
seee eeee emmm mmmm mmmm mmmm mmmm mmmm
|[-- ---- ][-- ---- ---- ---- ---- ----]
0011 1110 0100 1100 1100 1100 1100 1101
```

Числа с плавающей точкой

Представление числа:

$$X = (-1)^S \times 1.M \times 2^{E-E_0}$$

Значение $E_0 = 2^{p-1} - 1$.
Мы будем рассматривать примеры для типа `float` ($E_0 = 127$).

Пример 3: число
$\frac{1}{5} = (-1)^0 \times 1.10011001100110011001100110011\ldots \times 2^{124-127}$.

```
seee eeee emmm mmmm mmmm mmmm mmmm mmmm
|[-- ---- ][-- ---- ---- ---- ---- ----]
0011 1110 0100 1100 1100 1100 1100 1101
```

Специальные значения

Кроме чисел, в типе с плавающей точкой могут храниться специальные значения.

Знак	Экспонента	Мантисса	Значение
0	00..00	00..00	$+0$
0	00..00	00..01 ... 11..11	$+0.m \times 2^{-e_0+1}$
0	00..01 ... 11..10	XX..XX	$+1.m \times 2^{e-e_0}$
0	11..11	00..00	$+\infty$
0	11..11	00..01 ... 01..11	SNaN
0	11..11	10..00 ... 11..11	QNaN
1	00..00	00..00	-0
1	00..00	00..01 ... 11..11	$-0.m \times 2^{-e_0+1}$
1	00..01 ... 11..10	XX..XX	$-1.m \times 2^{e-e_0}$
1	11..11	00..00	$-\infty$
1	11..11	00..01 ... 01..11	SNaN
1	11..11	10..00 ... 11..11	QNaN

Специальные значения

Кроме чисел, в типе с плавающей точкой могут храниться специальные значения.

Знак	Экспонента	Мантисса	Значение
0	`00..00`	`00..00`	$+0$
0	`00..00`	`00..01` ... `11..11`	$+0.m \times 2^{-e_0+1}$
0	`00..01` ... `11..10`	`XX..XX`	$+1.m \times 2^{e-e_0}$
0	`11..11`	`00..00`	$+\infty$
0	`11..11`	`00..01` ... `01..11`	`SNaN`
0	`11..11`	`10..00` ... `11..11`	`QNaN`
1	`00..00`	`00..00`	-0
1	`00..00`	`00..01` ... `11..11`	$-0.m \times 2^{-e_0+1}$
1	`00..01` ... `11..10`	`XX..XX`	$-1.m \times 2^{e-e_0}$
1	`11..11`	`00..00`	$-\infty$
1	`11..11`	`00..01` ... `01..11`	`SNaN`
1	`11..11`	`10..00` ... `11..11`	`QNaN`

Примеры

Несколько простых упражнений.

Примеры

Несколько простых упражнений.

Пример 1:

```
volatile float a, b;
a = 0;
while (1)
{
  b = a + 1;
  if (a == b)
    break;
  a = b;
}
printf ("%f\n", a);
```

Примеры

Несколько простых упражнений.

Пример 1:

```
volatile float a, b;
a = 0;
while (1)
{
  b = a + 1;
  if (a == b)
    break;
  a = b;
}
printf ("%f\n", a);
```

Программа выведет число $16777216.000000 = 2^{24}$.

Примеры

Несколько простых упражнений.

Пример 2:

```
volatile float a, b;
a = 1E10;
b = 1E10 + 1000;
while (a < b)
  a = a + 1;
printf ("%f\n", a);
```

Примеры

Несколько простых упражнений.

Пример 2:

```
volatile float a, b;
a = 1E10;
b = 1E10 + 1000;
while (a < b)
  a = a + 1;
printf ("%f\n", a);
```

Это вечный цикл: увеличение `a` на единицу фактически не происходит.

Примеры

Несколько простых упражнений.

Пример 3:

```
volatile float a, b;
a = 1E10;
b = 1E10 + 10;
while (a < b)
  a = a + 1;
printf ("%f\n", a);
```

Примеры

Несколько простых упражнений.

Пример 3:

```
volatile float a, b;
a = 1E10;
b = 1E10 + 10;
while (a < b)
  a = a + 1;
printf ("%f\n", a);
```

Этот цикл не выполнится ни разу, так как $\mathtt{a} = \mathtt{b}$.
Будет выведено число 10000000000.000000.

Примеры

Несколько простых упражнений.

Пример 4:

```
volatile float a, b;
a = 1;
a += 1E10;
a -= 1E10;
printf ("%f\n", a);
```

Примеры

Несколько простых упражнений.

Пример 4:

```
volatile float a, b;
a = 1;
a += 1E10;
a -= 1E10;
printf ("%f\n", a);
```

После сложения на хранение единицы не хватит разрядов.
Будет выведено число `0.000000`.

Примеры

Несколько простых упражнений.

Пример 5:

```
volatile float a, b;
a = 1000;
a += 1E10;
a -= 1E10;
printf ("%f\n", a);
```

Примеры

Несколько простых упражнений.

Пример 5:

```
volatile float a, b;
a = 1000;
a += 1E10;
a -= 1E10;
printf ("%f\n", a);
```

После сложения не все разряды тысячи удастся сохранить. Будет выведено число `1024.000000`.

Всё.