

Практическая работа №3


«Выполнение арифметических операций над двоичными числами»

1. Цель работы:


- Научиться конвертировать числа между десятичной, двоичной и шестнадцатеричной системами счисления, использовать двоичные числа для арифметических операций и анализировать результаты с учетом разрядности и кодирования со знаком


2. Кодирование целых неотрицательных чисел

Широко известно, что вычислительная техника использует двоичное кодирование в ячейках фиксированной ширины для хранения и передачи самых разных типов данных. Наиболее «очевидными» в этом отношении являются целочисленные неотрицательные целые числа (`unsigned int` и прочие `unsigned`-типы в C/C++). В этом случае трактовка всех битов одинакова и сводится к двоичным цифрам от 0 до 2^{K-1} (от $00...00_2$ до $11...11_2$), трактуя все биты как двоичные цифры.

 Формулировка «двоичное кодирование в ячейках *фиксированной ширины*» подразумевает, что в пределах этих ячеек коды «циклически» сменяют друг друга. Например, для 16-битного слова за числом $FFFF_{16} = 1111.1111.1111.1111_2 = 65535_{10}$ «следует» 0, а не 65536 (потому что полная запись числа $65536_{10} = 1.0000.0000.0000.0000_2$ (точки для наглядности)).

На уровне процессора этот эффект отслеживается **флагом переноса (CF)**, который сразу после выполнения команды, приведшей к переносу, переключается из 0 в 1, и на языке ассемблера можно «учесть» этот флаг например для того «перенести» 1 в соседнюю, условно более старшую ячейку (или из соседней ячейки, если речь идет про вычитание из неотрицательного 0 неотрицательной 1 и записи в неотрицательную ячейку).

 **Задание 1.** Попробуйте в любом языке программирования (рекомендуется C/C++, но можно любой) узнать «пределы» ширины ячейки *неотрицательного* числа последовательным увеличением. Обратите внимание, что речь идет только про **беззнаковые** целые: *числа со знаком имеют свои нюансы в обращении (см. п. 3)!*

 **Совет:** прежде чем выполнять задания, если не уверены, прочитайте и ознакомьтесь с остальными пунктами методички.

Примечание. В некоторых языках программирования этого предела нет, так как реализована поддержка длинной арифметики (то есть, реализованы числа на произвольном количестве ячеек), в таком случае «докажите» что

предела переноса нет (точнее, что перенос учитывает автоматически), по крайней мере, на примере числа большего, чем 2^{128} .

3. Кодирование целых чисел со знаком

Каким образом можно закодировать число со знаком? Если бы речь шла про троичную симметричную систему, в которой знак заложен в цифру разряда, то можно было бы сказать что это реализуется «автоматически». Однако в современных компьютерах используется двоичная система счисления и двоичное кодирование, а это значит, что за знак «должен» отвечать отдельный бит ячейки.

3.1 Прямой код

Возьмем самый старший бит ячейки и будет кодировать им знак числа.


Например, для однобайтовых чисел $\underline{1}000.0001_{\text{п.к.}} - 0000.0001_2 = -1_{10}$.

Почему этот код не очень удобен для нужд компьютеростроения, хотя он очень «интуитивен» для нас с вами: дело в том, что в таком случае сложение и вычитание чисел придется реализовывать отдельно, а также всегда отдельно учитывать знак числа.

3.2 Обратный код появился из идеи, что раз инвертирование старшего бита инвертирует знак, то можно было бы инвертировать и все прочие биты тоже: $\underline{1}111.1110_{\text{о.к.}} = -0000.0001_2 = -1_{10}$.

Обратный код лишь немного улучшил ситуацию, ведь теперь, чтобы инвертировать знак числа, не требовалось отдельно реализовать инвертор для знака числа, было достаточно использовать побитовое НЕ, доступное в большинстве архитектур.

3.3 Дополнительный код появляется «увеличением» на 1 относительно обратного. Его основная идея: раз ячейка дает нам циклическое пространство чисел, так давайте воспользуемся этой «циклическостью»! В этой системе кодирования вычитание реализуется сложением, а это значит, что можно существенно упростить микросхемы, увеличив их надежность и дешевизну (либо мощность).

 Проще всего понять идею дополнительного кода на примере того, как в нем записывается число -1, представив себе его как результат вычитания единицы из 0 (как будто мы вычитаем не из 0, а из 256 - для ячейки в один байт шириной):

$$\begin{array}{r} (1)0000.0000 \text{ как } 0, 256, 512, \dots, 256n + 0 \\ - \quad 0000.0001 \text{ (минус один)} \\ \hline 1111.1111_{\text{д.к.}} = 1111.1110_{\text{о.к.}} + 1 = -1_2 = -1_{10} \end{array}$$

На всякий случай покажем в таблице все значения, чему соответствует каждая комбинация для дополнительных кодов:

| Комбинация | ≥ 0 | Д.К. | О.К. | П.К. |
|------------|----------|------|------|------|
| 0000 | 0 | 0 | 0 | 0 |
| 0001 | 1 | 1 | 1 | 1 |
| 0010 | 2 | 2 | 2 | 2 |
| 0011 | 3 | 3 | 3 | 3 |
| 0100 | 4 | 4 | 4 | 4 |
| 0101 | 5 | 5 | 5 | 5 |
| 0110 | 6 | 6 | 6 | 6 |
| 0111 | 7 | 7 | 7 | 7 |
| 1000 | 8 | -8 | | |
| 1001 | 9 | -7 | | |
| 1010 | 10 | -6 | | |
| 1011 | 11 | -5 | | |
| 1100 | 12 | -4 | | |
| 1101 | 13 | -3 | | |
| 1110 | 14 | -2 | | |
| 1111 | 15 | -1 | | |

Процессор отслеживает состояние знаковой ячейки не только посредством уже знакомого нам Carry flag, но и посредством Overflow flag. Флаг переполнения активируется, когда представление числа не уместилось в К-1 битов ячейки и «коснулось» в том числе старшего бита знака (то есть, когда бит знака предположительно неверный).



Задание: заполните остаток таблицы для ячейки шириной 4 бита.



Используя любой язык программирования, имеющий поддержку как чисел со знаком, так и исключительно беззнаковых чисел (рекомендуется C/C++), попробуйте определить, какой код из представленных использует Ваш компьютер.



Контрольный вопрос: подумайте и скажите, в каком коде (или каких кодах) есть два представления числа 0, как +0 и -0? Какими двоичными последовательностями (например, для ячейки 16 бит) может быть представлен ноль?



Используя любой язык программирования, добейтесь вызова Overflow flag. Это можно сделать, последовательно увеличивая знаковую ячейку до отрицательного числа, либо последовательно вычитая из отрицательного числа, пока не будет положительное. Рекомендуется C/C++ для выполнения задания, но допускается любой язык, где числа могут храниться как целые, и в то же время хранятся в ячейке фиксированного размера (без длинной арифметики).



Для самопроверки всех задач можно использовать сайт <https://carlosrafaelgn.com.br/Asm86/>, содержащий онлайн-эмулятор архитектуры x86, правда, придется разобраться в простейших командах ассемблера x86, таких как mov, jmp (аналог goto), и прочих. Вместо переменных можно использовать регистры процессора, например, 32-битные eax, ebx, ecx. Можете попробовать начать с такой «программы»:

```
mov eax, 10
mov ebx, 2300000000
cycle_again:
nop
add eax, ebx

jmp cycle_again
```

4. Кодирование дробных чисел по стандарту IEEE 754

Как кодировать вещественные числа? Для того, чтобы кодировать числа с фиксированной запятой (например, количество рублей с учетом копеек, которые всегда не выйдут за пределы сотых долей рубля¹) ничего «нового» относительно целых чисел со знаком не нужно.

Однако, если мы решаем физические или, например, геометрические задачи, решаем задачи машинного обучения, то оказывается, что нам необходимо число с «подвижной» точностью, а точнее, плавающей точностью. Практика требует от нас уметь работать как с очень большими числами на сравнительно низкой точности, например, с числом $6,02214076 \cdot 10 \times 10^{23}$, так и с сравнительно малыми числами, например, $1,054571817 \times 10^{-34}$ (обе примера реально используются в физике и химии; возможно, вы сталкивались с ними на других предметах или даже в школе).

Если бы мы работали с последовательностями битов произвольной длины, без ячеек, то такой проблемы бы не было², достаточно было бы просто «выровнять» последовательность относительно 0, добавив «нулей» с конца записи или с начала, и учитывая их при умножении или сложении.

¹ на самом деле, десятитысячных долей рубля, т.к. копейка также делима до сотых долей по закону, но на самом деле никак не более того: одна миллионная рубля уже не существует в принципе

² проблемы были бы, но другие. Кроме того, возможно лучше было бы использовать ранее упомянутую троичную симметричную систему с тритами, тогда бы избавились от необходимости отдельно считать округление чисел

Но для существующих архитектур обычно применяется иной способ кодирования: было предложено разбить все биты на три «группы», в соответствии с моделью «научной экспоненциальной» записи числа, но не в десятичной системе счисления, а в двоичной нормализованной:

$$1,75_{10} = + 1,11_2 \times 2^0$$

Дробные цифры двоичной системы счисления работают по тому же принципу, что и обычные, просто за знаком десятичной запятой вместо умножения на 2 каждый следующий разряд делится на 2. Если есть затруднения в переводе чисел в двоичной системе счисления, лучше потренироваться.

Знак числа кодируется одним битом (это первая группа битов). В данном случае это 0

Далее кодируют целочисленный порядок числа (это показатель степени при двойке), для чего используют дополнительный код (да, допускается и отрицательный показатель степени, но строго целочисленный!), в данном случае это тоже ноль: 00...00.

Наконец, чтобы закодировать значащие цифры, используют оставшиеся биты, называя это мантиссой. Поскольку всякое число полагается записывать нормализованным (то есть, с ведущей значащей цифрой), а у двоичной системы всего две цифры, но на первой «единице» можно сэкономить, что и делают. В данном примере это $1,11000..._2 = \text{то есть } 1,1100....$

Для 32-битного числа (float для C++ в x86_64, x86) итого имеем:

0000 0000.01100 0000.0000 0000.0000 0000

(Точками разграничены байты, а пробелами - нибблы в пределах байта)

Большее количество примеров доступно на [сайте](#) (немного устаревший, но информации много).



Задание: нормализовать число из варианта (см п.5), перевести в двоичную систему счисления и записать число в системе счисления в соответствии с вариантом из п.5 в 16-ричной точности хотя бы четыре значащие десятичные цифры (допускается больше, меньше нельзя). Для физических величин следовать СИ.



Вариант выполнения задания: можно попробовать перевести число используя разбор числа по битам и используя каламбуры типизации (такие как union или переводы типов данных в языке C++). Использование сторонних калькуляторов - строго в порядке самопроверки ответа.

6. Варианты для самостоятельной работы

| Номер варианта | Число |
|-------------------|-------|
| 1 | Пи |

| | |
|----|---|
| 2 | Эйлера |
| 3 | Авогадро |
| 4 | Постоянная Планка приведенная |
| 5 | Гравитационная постоянная G |
| 6 | Скорость света в вакууме |
| 7 | Температура абсолютного нуля (в Цельсиях) |
| 8 | Золотое сечение |
| 9 | Температура плавления алюминия |
| 10 | Текущее атмосферное давление в Паскалях |
| 11 | Среднее расстояние от Земли до Луны (км) |
| 12 | Температура испарения кислорода (в Цельсиях) |
| 13 | Ускорение материальной точки в момент столкновения с поверхностью, м/с ² |
| 14 | Сумма всех натуральных чисел (не бесконечность) |
| 15 | Атомная масса урана-238 |
| 16 | Количество комбинаций кубика Рубика |
| 17 | Средняя скорость улитки (св.лет/сек) |
| 18 | Постоянная Капрекара |
| 19 | Магнитный момент электрона |
| 20 | 0 F° (в градусах Цельсия) |
| 21 | ВВП РФ на душу населения (в USD) |
| 22 | Доля побед по отношению ко всем номинациям на награду у фильма Звездные войны IV: Новая надежда |
| 23 | Самая высокая погодная дневная температура на Земле |
| 24 | -42,0 |
| 25 | Планковская длина |
| 26 | Общее количество натуральных чисел |
| 27 | Планковское время |
| 28 | Постоянная Фарадея |

| | |
|----|---|
| 29 | удельный заряд электрона |
| 30 | Газовая постоянная |
| 31 | Постоянная Ридберга |
| 32 | Волновое сопротивление вакуума |
| 33 | Количество ячеек в машине Тьюринга |
| 34 | Предельное количество действий в машине Поста |
| 35 | Результат деления 0/0 |

6. Вопросы для самостоятельного контроля

1. Что такое дополнительный код?
2. Как перевести число из двоичной системы в шестнадцатеричную систему?
3. Как определить знак числа?
4. В каких случаях флаги состояний процессора CF и OF будут совпадать?

7. Список литературы для самостоятельного изучения

1. Хеннесси, Дж. Л., Паттерсон, Д. А. Компьютерная архитектура: количественный подход. — 5-е изд. — М.: Вильямс, 2016. — 944 с.
2. Таненбаум, Э. Архитектура компьютера. Структурный подход. — 5-е изд. — СПб.: Питер, 2013. — 832 с.
3. Архитектура вычислительных систем [Электронный ресурс]: учебное пособие – Эл. изд. - Электрон. текстовые дан. (1 файл pdf: 77 с.). - Грейбо С.В., Новосёлова Т.Е., Пронькин Н.Н., Семёнычева И.Ф. 2019