Содержание

[1 Цель работы 3](#_Toc511342496)

[2 Постановка задачи 4](#_Toc511342497)

[3 Краткие теоретические сведения 5](#_Toc511342498)

[3.1 Представление чисел в формате с плавающей точкой 5](#_Toc511342499)

[4 Арифметические операции над числами с плавающей точкой 11](#_Toc511342500)

[4.1 Операции над числами 11](#_Toc511342501)

[4.2 Сложение и вычитание 11](#_Toc511342502)

[4.3 Умножение и деление 13](#_Toc511342503)

[5 Демонстрация работы программы 15](#_Toc511342504)

[6. Вывод 21](#_Toc511342505)

[Приложение. Исходный код программы 22](#_Toc511342506)

# Цель работы

Разработать АЛУ, производящий арифметические операции над числами с плавающей точкой.

# Постановка задачи

Написать программу эмулятора АЛУ, реализующего операции сложения, вычитания, умножения, деления с плавающей точкой над двумя введенными числами, с возможностью пошагового выполнения алгоритмов.

# Краткие теоретические сведения

## Представление чисел в формате с плавающей точкой

В формате с фиксированной точкой, в частности в дополнительном коде, можно представлять положительные и отрицательные числа в диапазоне, симметричном на числовой оси относительно точки 0. Расположив воображаемую разделяющую точку в середине разрядной сетки, можно в этом формате представлять не только целые, но и смешанные числа, а также дроби.

Однако такой подход позволяет представить на ограниченной разрядной сетке множество вещественных чисел в довольно узком диапазоне. Нельзя представить очень большие числа или очень маленькие. При выполнении деления двух больших чисел, как правило, теряется дробная часть частного.

При работе в десятичной системе счисления ученые давно нашли выход из положения, применяя для представления числовых величин так называемую научную нотацию. Так, число 976 000000 000 000 можно представить в виде 9.76x1014, а число 0,000000 000 000 0976 - в виде 9.76x10-14. При этом, фактически, разделительная точка динамически сдвигается в удобное место, а для того чтобы "уследить" за ее положением в качестве второго множителя - характеристики, - используется степень числа 10 (основания характеристики). Это позволяет с помощью небольшого числа цифр (т.е. чисел с ограниченной разрядностью) с успехом представлять как очень большие, так и очень малые величины.

Этот же подход можно применить и в двоичной системе счисления. Число можно представить в виде

*±SxB±E*

Компоненты такого представления можно сохранить в двоичном слове, с, стоящем из трех полей:

        поле знака числа (плюс или минус)

        поле мантиссы S;

        поле порядка E;

Основание B подразумевается неявно и не сохраняется.

Принципы представления лучше пояснить на примерах. На рис. 1 схематически показан 32-х разрядный формат с плавающей точкой. В крайнем левом бите слова хранится знак числа (0 – положительное, 1- отрицательное). В следующих 8 битах хранится значение порядка. Для представления порядка используется так называемый смещенный формат. Для получения действительного двоичного кода порядка из значения, сохраняемого в этом поле нужно вычесть фиксированное смещение. Как правило, смещение равно (2k-1-1), где k – разрядность поля порядка. В данном случае k = 8, и в поле порядка можно представить коды в диапазоне от 0 до 255. Если принять значение смещения 127, то действительное значение порядка чисел, представленных в таком формате может находится в интервале от -127 до +128. В данном примере считается, что основание характеристики совпадает с основанием системы счисления и равно 2.

Если интерпретировать смещенное ставление как обычный формат без знака, то отношение старшинства между кодами сохраняется, т.е. наибольшее значение в обоих случаях есть 1111, а наименьшее — 0000. Этого не наблюдается при представлении чисел в прямом или дополнительном кодах. Преимущество смещенного представления порядка в формате числа с плавающей точкой в том, что результат сравнения двух неотрицательных вещественных чисел будет таким же, как и результат сравнения их кодов, рассматриваемых как целые числа без знака.

Последнее поле в слове (23 бит) отводится для хранения значения мантиссы S. Теперь хочу обратить ваше внимание на следующий нюанс. Любое число можно представить в форме с плавающей точкой множеством способов. Так, приведенные ниже формы представления эквивалентны, если считать, что мантисса выражена в двоичной системе счисления:

*0.110х25*

*110х22*

*0.0110х26*

Для упрощения алгоритмов выполнения арифметических операций обычно принято нормализовать мантиссу. Нормализованная мантисса числа, отличного от нуля, имеет вид

0.1bbb...bх2±Е,

где b представляет произвольную двоичную цифру (0 или 1). Это означает, что старший (левый) значащий разряд кода мантиссы всегда равен 1. Но если он всегда равен 1, его нет смысла хранить в составе числа, а можно просто учитывать этот факт при выполнении операций. Таким образом, в 23-битовом поле фактически хранится 24-разрядный код мантиссы, значение которой может был в диапазоне от 0.510 до 1.0.

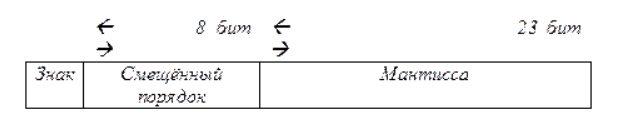


Рисунок 1. Представление вещественных чисел

Ниже приведены примеры чисел в формате с плавающей точкой.

|  |  |
| --- | --- |
| *0.11010001х210100=* | *0 10010011 10100010000000000000000* |
| *-0.11010001х210100 =* | *1 10010011 10100010000000000000000* |
| *0.11010001х2-10100 =* | *0 01101011 10100010000000000000000* |
| *-0.11010001х2-10100 =* | *1 01101011 10100010000000000000000* |

Особенности:

        знак сохраняется в старшем бите слова;

        первый разряд мантиссы всегда равен 1, и в поле мантиссы не хранится;

        к действительному значению порядка прибавляется смещение 127 и в поле порядка хранится эта сумма;

        основание характеристики равно 2.

Если в слове такой же длины хранить целые числа в дополнительном коде, то диапазон представления будет охватывать 232 чисел от 2-31 до 231-1 включительно. В формате с плавающей точкой с распределением полей, как показано на рис. 8.10, можно хранить:

отрицательные числа от -(1-2-24)х2128 до -0.5х2-127,

положительные числа от 0.5х2-127 до (1-2-24)х2128.

Пять областей на числовой оси не включены в диапазон представления:

        отрицательные числа, меньшие, чем -(1-2-24)х2128, эта область именуется отрицательной областью переполнения (negative overflow);

         отрицательные числа, большие, чем -0.5х2-127, эта область именуется отрицательной областью потери значимости (negative underflow);

        нуль;

        положительные числа, меньшие, чем 0.5х2-127, эта область именуется положительной областью потери значимости (positive underflow);

        положительные числа, большие, чем (1-224)х2128, эта область именуется положительной областью переполнения (positive overflow).

Строго говоря, описанное выше представление не позволяет хранить число 0, но на практике код, состоящий из нулей во всех разрядах, считается допустимымипредставляет число 0. В области переполнения можно попасть в том случае, если результат арифметической операции имеет абсолютную величину, превышающую ту, которая представляется нормализованным числом с порядком 128, а в область потери значимости – если результат операции имеет очень маленькую абсолютную величину. Потеря значимости как правило не представляет особой проблемы, поскольку в таких случаях с достаточной точностью результат можно считать равным 0.

Обращаю ваше внимание на то, что формат с плавающей точкой не позволяет представить больше отличающихся друг от друга числовых величин. Их по-прежнему 232 (для слова в 32 бита). Интервал между соседними числами – переменный и зависит от абсолютной величины числа.. Фактически результаты округляются с точностью, определяемой имеющейся разрядной сеткой.

В рассматриваемом примере формата под порядок отводится 8 бит, а под мантиссу — 23 бит. Если увеличить длину поля порядка, диапазон представления расширится, но при этом придется сократить поле мантиссы, а значит точность представления снизится, т.е. уменьшится "плотность" размещения представляемых величин на числовой оси. Единственный способ увеличить диапазон, и точность представления — расширить разрядную сетку. В большинстве компьютеров имеются два формата с плавающей точкой: обычный и удвоенной точности. Например, типовой формат обычной точности занимает 32-битовое слово, а формат удвоенной точности занимает 64 бит.

3.2 Стандарт IEEE формата с плавающей точкой

Для унификации формата представления чисел с плавающей точкой, что явля­ется необходимым условием переносимости программного обеспечения, Институтом инженеров по электротехнике и радиоэлектронике IEEE разработан стандарт 754 . В последнее десятилетие практически все процессоры и арифметические сопроцессоры проектируются с учетом требований этого стандарта.

Стандарт специфицирует два варианта формата: 32-битовый — обычной точности представления и 64-битовый — удвоенной точности представления. В первом формате поле порядка занимает 8 бит, а во втором -11 бит. Стандарт регламентирует использование числа 2 в качестве неявно заданного значения основания характеристики. Помимо основных, в стандарте предусмотрены два расширенных варианта форматов обычной и удвоенной точности, конкретная спецификация которых зависит от реализации вычислительной системы. Расширенные форматы позволяют включать дополнительные биты в поле порядка (расширение диапазона представления) в поле мантиссы (повышение точности представления). Расширенные форматы предназначаются для промежуточных вычислений. За счёт повышения точности снижается вероятность появления ошибок округления, а при расширении диапазона снижается вероятность появления ошибки переполнения.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Параметр* | *Формат* | | | |
|  | *Обычная точность* | *Расширенный обычной точности* | *Удвоеннаяточность* | *Расширенныйудвоенной точности* |
| *Размер слова (бит)* | *32* | *≥43* | *64* | *≥79* |
| *Поле порядка (бит)* | *8* | *≥11* | *11* | *≥15* |
| *Смещение порядка* | *127* | *Не регламентируется* | *1023* |  |
|  |
| *Максимальное значение порядка* | *127* | *≥1023* | *1023* | *≥16383* |
| *Минимальное значениепорядка* | *-126* | *≤-1022* | *-1022* | *≤-16382* |
|  |  |  |  |
| *Диапазон представления(по основанию 10)* | *10-38, 10+38* | *Не регламентируется* | *10-308, 10+308* | *Не регламентируется* |
| *Поле мантиссы (бит)* | *23* | *≥31* | *52* | *≥63* |
| *Количество значенийпорядка* | *254* | *Не регламентируется* | *2046* | *Не регламентируется* |
|  |  |
| *Количество значениймантиссы* | *223* | *Не регламентируется* | *252* | *Не регламентируется* |
|  |  |
| *Количество отличающихся представимых* *величин* | *1.98х231* | *Не регламентируется* | *1.99х263* | *Не регламентируется* |
|  |  |
|  |  |  |  |

Таблица 1. Параметры форматов, регламентированные стандартом IEEE 754

Не включая неявный старший разряд, равный 1

Не все двоичные комбинации в форматах, регламентированных стандартом, интерпретируются обычным способом (табл. 1). Крайние значения порядка — все нули (значение 0) и все единицы (значение 255 в формате обычной точности и 2047 в формате удвоенной точности) — представляют особые величины. Стандартом предусматривается представление в заданном формате следующих классов числовых величин:

        При значениях кода в поле порядка в интервале от 1 до 254 в формате обычной точности и от 1 до 2046 в формате удвоенной точности представляются нормализованные вещественные числа. Используется смещенный формат представления порядка и, следовательно, интервал возможных значений порядка простирается от -126 до +127 в формате обычной точности и от -1022 до +1023 в формате удвоенной точности. Нормализованная мантисса включает бит, равный 1, слева от разделительной точки, который не хранится в поле мантиссы. Следовательно, в формате обычной точности представляется 24-разрядная мантисса, а в формате удвоенной точности — 53-разрядная мантисса (в стандарте используется термин fraction — дробная часть).

        Нулевое значение в поле порядка совместно с нулевым значением в поле мантиссы представляет значение +0 или -0 в зависимости от кода в бите знака мантиссы.

        Заполнение поля порядка кодом 1 совместно с нулевым кодом в поле мантиссы представляет величины +∞ и -∞ в зависимости от кода в бите знака мантиссы. Таким образом, формат позволяет представить на ограниченной разрядной сетке неограниченно большую числовую величину. Пользователь (конструктор вычислительной системы) должен решить, трактовать ли переполнение как ошибку или обрабатывать в программе бесконечно большие числовые величины.

        Нулевое значение в поле порядка совместно с ненулевым значением в поле мантиссы представляет ненормализованные числа. В этом случае считается, что неявный старший разряд мантиссы равен 0, и значение порядка равно -126 в формате обычной точности или -1022 — в формате удвоенной точности. Код в знаковом бите мантиссы по-прежнему определяет знак числа.

        Заполнение поля порядка кодом 1 совместно с ненулевым кодом в поле мантиссы представляет так называемые необычные числовые типа NaN - Not a Number).

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | *Обычная точность (32 бит)* | | | | *Удвоенная точность (64 бит)* | | | |
|  | *Знакмантисс­­сы* | *Смеще­нный*  *порядок* | *Ман­ти­сса* | *Значе­ние* | *Знак мантис­сы* | *Смеще­н­ный порядок* | *Мантис­са* | *Значе­ние* |
| Положитель­ный нуль | *0* | *0* | *0* | *0* | *0* | *0* | *0* | *0* |
| Отрицательный  нуль | *1* | *0* | *0* | *-0* | *1* | *0* | *0* | *-0* |
| Плюс  бесконечность | *0* | *255 (все единицы)* | *0* | *+ ∞* | *0* | *2047 (все единицы)* | *0* | *+ ∞* |
| Минус бесконечность | *1* | *255 (все единицы)* | *0* | *-∞* | *1* | *2047 (все единицы)* | *0* | *-∞* |
| Простое значение  типа NaN | *0 или 1* | *255 (все*  *единицы)* | *≠0* | *NaN* | *0 или 1* | *2047 (все единицы)* | *≠0* | *NaN* |
| Сигнализирующеезначение типа NaN | *0 или 1* | *255 (все единицы)* | *≠0* | *NaN* | *0 или 1* | *2047 (все единицы)* | *≠0* | *NaN* |
| Положительное нормализованное число,  отличное от 0 | *0* | *0<е<255* | *f* | *2e-127(1.f)* | *0* | *0<е<2047* | *f* | *2e-1023(1.f)* |
| Отрицательное нормализованное число,  отличное от 0 | *1* | *0<е<255* | *f* | *-2e-127(1.f)* | *1* | *0<е<2047* | *f* | *-2e-1023(1.f)* |
| Положительное ненормализованное число | *0* | *0* | *f≠0* | *2e-126(0.f)* | *0* | *0* | *f≠0* | *2e-1022(0.f)* |
| Отрицательноененормализован­ное число | *1* | *0* | *f≠0* | *-2e-126(0.f)* | *1* | *0* | *f≠0* | *-2e-1022(0.f)* |

Таблица 2. Представление величин в формате с плавающей точкой в соответствии со стандартом IEEE 754

# Арифметические операции над числами с плавающей точкой

## Операции над числами

Таблица.3 Арифметические операции над числами в формате с плавающей точкой.

*Примеры:*

*X = 0.3x102 = 30;*

*Y = 0.2x103 = 200;*

*X + Y = (0.3x102-3 + 0.2)x103 = 0.23x103 = 230;*

*X - Y = (0.3x102-3 - 0.2)x103 = (-0.17)x103 = -170;*

*X x Y= (0.3x0.2)x102+3 = 0.06x105 = 6000;*

*X / Y = (0.3/0.2)x102-3 = 1.5x10-1 = 0.15*

При сложении и вычитании необходимо предварительно уравнять порядки операндов, что требует сдвига положения разделительной точки в мантиссе. Операции умножения и деления уравнивания порядков операндов не требуют.

При выполнении операций могут возникнуть следующие особые ситуаци:

        **Переполнение порядка.**Положительный порядок результата превышает максимальное значение, предусмотренное форматом. В некоторых системах такой результат может трактоваться как величина +∞ или -∞.

        **Потеря значимости порядка.**Отрицательный порядок результата меньше минимального значения, допускаемого принятым форматом (например, получен порядок -200, который меньше разрешенного -127). Это означает, что получен очень малый результат, который можно считать равным нулю.

        **Потеря значимости мантиссы.**В процессе уравнивания порядков мантисса сдвигается настолько сильно вправо, что старший значащий ее разряд выходит за пределы разрядной сетки.

        **Переполнение мантиссы.**При сложении мантисс с одинаковыми знаками возможно появление переноса из старшего разряда.

## Сложение и вычитание

Алгоритмы выполнения операций сложения и вычитания в формате с плавающей точкой сложнее, чем аналогичные алгоритмы для чисел в формате с фиксированной точкой. Связано это, в первую очередь, с необходимостью выравнивания порядков операндов. Алгоритм включает четыре основных этапа.

1.     Проверка на нуль.

2.     Сдвиг мантисс для выравнивания порядков.

3.     Суммирование или вычитание мантисс.

4.     Нормализация результата.

Блок-схема типового алгоритма представлена на рис. 9. Детальный пошаговый анализ этого алгоритма покажет, какие функции используются при выполнении операций сложения и вычитания чисел в формате с плавающей точкой. В дальнейшем для определенности будем считать, что используется формат, регламентированный стандартом IEEE 754. Перед началом выполнения операций операнды должны быть помещены в регистры АЛУ. Если в используемом формате с плавающей точкой предполагается неявный старший разряд мантиссы, этот разряд должен быть в явном виде включен в регистры операндов, и все операции с ним в дальнейшем будут проводиться точно так же, как и с остальными разрядами мантиссы.

Поскольку операции сложения и вычитания отличаются только тем, что при вычитании предварительно изменяется знак второго операнда (вычитаемого), эта операция включена в ветвь ВЫЧИТАНИЕ на схеме алгоритма, после чего обе ветви сливаются. Далее анализируется, не равен ли один из операндов нулю. Если это так, то результат — значение второго операнда.

Следующий этап — изменение кодов операндов таким образом, чтобы значения их порядков стали равны. Чтобы понять, зачем это нужно, рассмотрим следующий пример в десятичной системе счисления:

*(123х100) + (456х10-2).*

Очевидно, что нельзя просто сложить мантиссы этих двух чисел. Сначала нужно выровнять разрядные сетки обеих мантисс так, чтобы соответственные разряды (разряды с равным весом) занимали одинаковые позиции, т.е. цифра 4 второго числа находилась в той же позиции, что и цифра 3 первого числа. При этом порядки обоих чисел будут равны. Равенство порядков и есть, с точки зрения математики, условие, позволяющее складывать мантиссы обоих чисел в такой форме представления. Следовательно,

*(123х100) + (456х10-2) - (123х100) + (4.56х100) = 127.56х100.*

Выравнивание выполняется за счет сдвига мантиссы меньшего числа вправо или мантиссы большего числа — влево. Поскольку в любом варианте теряются цифры операнда, выполняется сдвиг мантиссы меньшего числа вправо, что приводит к утере ее младших разрядов. Одновременно со сдвигом мантиссы вправо порядок меньшего числа увеличивается. Сдвиги выполняются до тех пор пока значения порядков обоих чисел не станут равны.

После того, как порядки будут выровнены, наступает этап сложения мантисс с учетом их знаков. Поскольку слагаемые могут иметь разные знаки, их алгебраическая сумма может оказаться равной нулю. Не исключено и появление переполнения — переноса из старшего разряда суммы. В этом случае необходимо выполнить дополнительный сдвиг результата вправо и одновременно увеличить на 1 значение порядка. Но тогда возможно появление переполнения порядка, которое расценивается как аварийная ситуация и влечет за собой прекращение операции.

После сложения мантисс наступает этап нормализации результата. Нормализация представляет собой серию сдвигов кода мантиссы влево (в сторону старших разрядов) с одновременным уменьшением значения порядка до тех пор, пока значение старшей цифры мантиссы не станет отличным от нуля. Я специально оговариваю цифры, поскольку в случае, когда основанием характеристики является число 16, шестнадцатеричной цифрой мантиссы будет служить 4-разрядный двоичный код. Последняя операция – округление результата.

## Умножение и деление

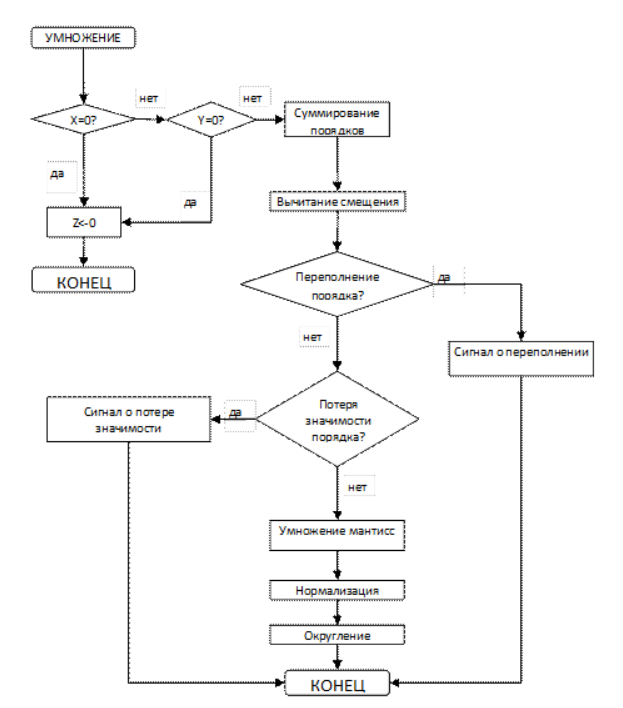
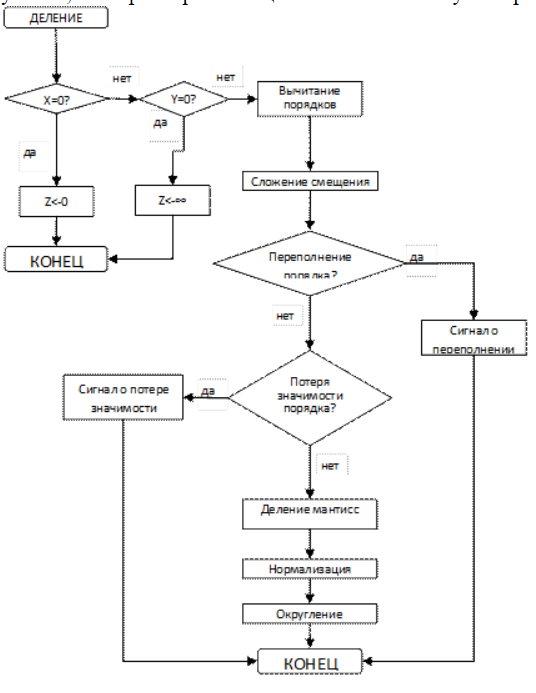
При работе с числами в формате с плавающей точкой алгоритмы умножения и деления оказываются проще алгоритмов сложения и вычитания.

Схема алгоритма умножения чисел в формате с плавающей точкой.

Рассмотрим алгоритм умножения. Сразу же после начала операции проверяется, не равен ли нулю один из сомножителей. Если это так, то произведение также будет равно нулю. Следующий шаг — суммирование порядков. Поскольку, как правило, для хранения порядков используется смещенное представление, при суммировании двух смещенных представлений результат будет смещен дважды. Поэтому после суммирования кодов порядков из суммы вычитается значение смещения. При суммировании может возникнуть как переполнение порядка, так и потеря значимости. В обоих случаях формируется соответствующий сигнал. Если порядок произведения не выходит из диапазона, определенного форматом, далее перемножаются мантиссы сомножителей с учетом их знаков. Умножение мантисс выполняется по тому же алгоритму, что и умножение целых чисел в прямом коде, т.е. фактически перемножаются числа без знака, а затем произведению приписывается знак "плюс" или "минус" в зависимости от сочетания знаков сомножителей. Произведение мантисс имеет разрядность, вдвое большую, чем каждый из сомножителей. Лишние младшие разряды отбрасываются при округлении.

После того как будет получено произведение мантисс, результат нормализуется и округляется. Эти операции выполняются так же, как и при сложении или вычитании. Необходимо учесть, что при нормализации может возникнуть переполнение или потеря значимости порядка.

Теперь рассмотрим алгоритм деления. Как и ранее, первый этап — анализ операндов на равенство нулю. Если нулю равно делимое, то результату сразу присваивается значение 0. Если же нулю равен делитель, то в зависимости от конкретной реализации АЛУ результату может быть присвоено значение "бесконечность" с соответствующим знаком или сформирован сигнал арифметической ошибки.

Следующий этап — вычитание кода порядка делителя из кода порядка делимого. При этом получится несмещенный код разности, который нужно скорректировать — сложить с кодом смещения. После завершения операций с порядком результата проверяется, не возникло ли переполнение порядка или потеря значимости.

Следующий этап — деление мантисс. За ним следуют обычные операции нормализации и округления.

# Демонстрация работы программы

Основные команды в данном эмуляторе:

mov – помещение в регистр значения

add – добавление значения к регистру

sub – вычитание из регистра значения

mul – умножение регистра на значение

div – деление регистра ax на значение

state – просмотр состояния регистров

exit – выход из программы

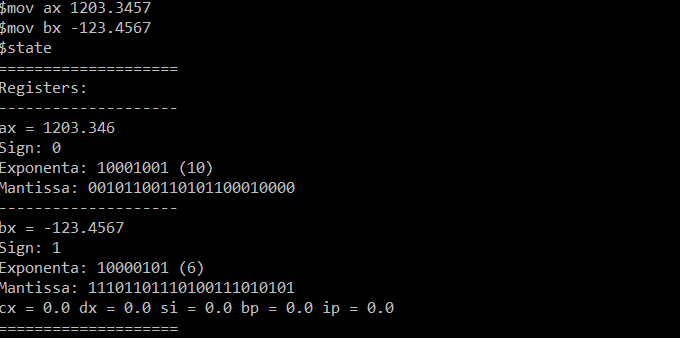


Рис 1. Помещение значение в регистры.

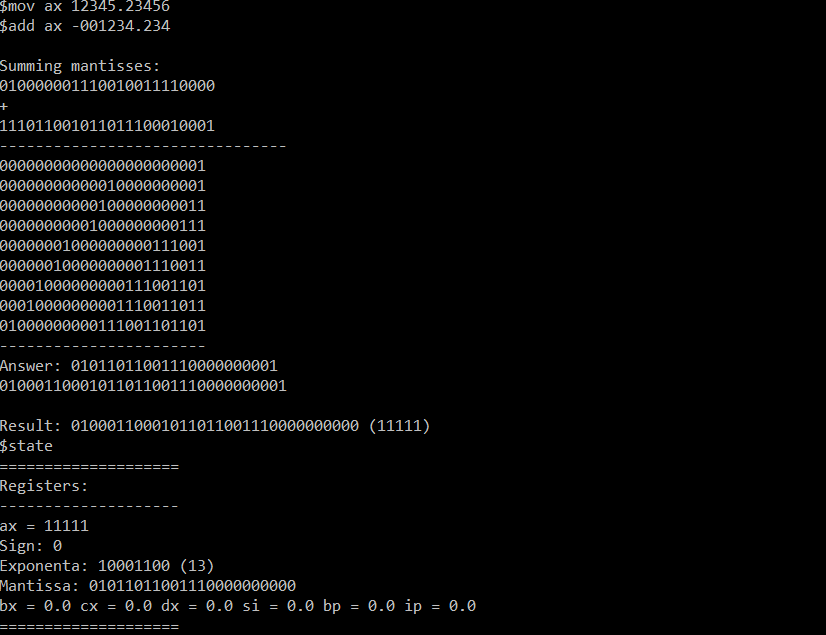


Рис 2. Сложение

(также есть ввод с нулями вначале числа)

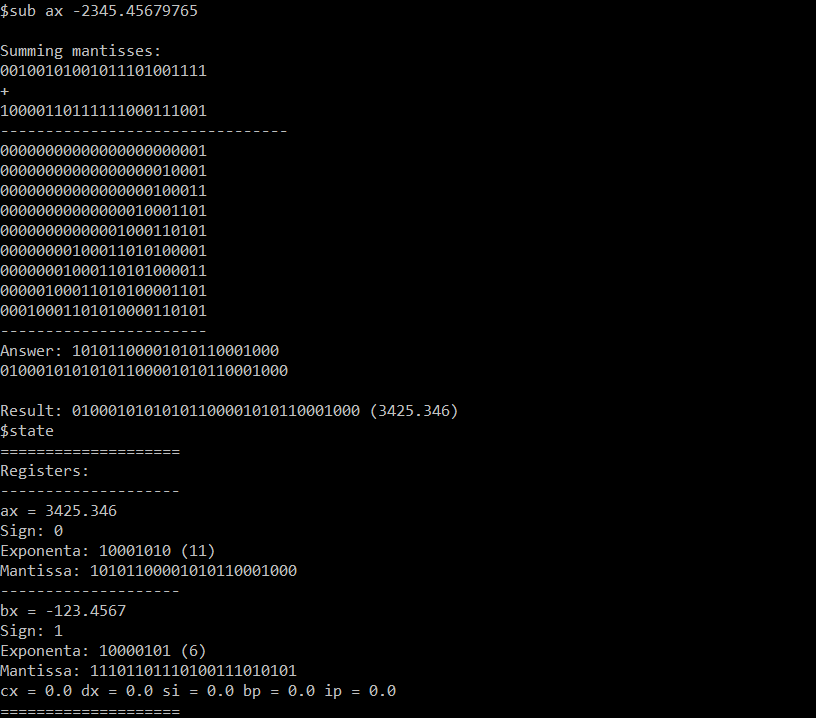


Рис 3. Вычитание чисел

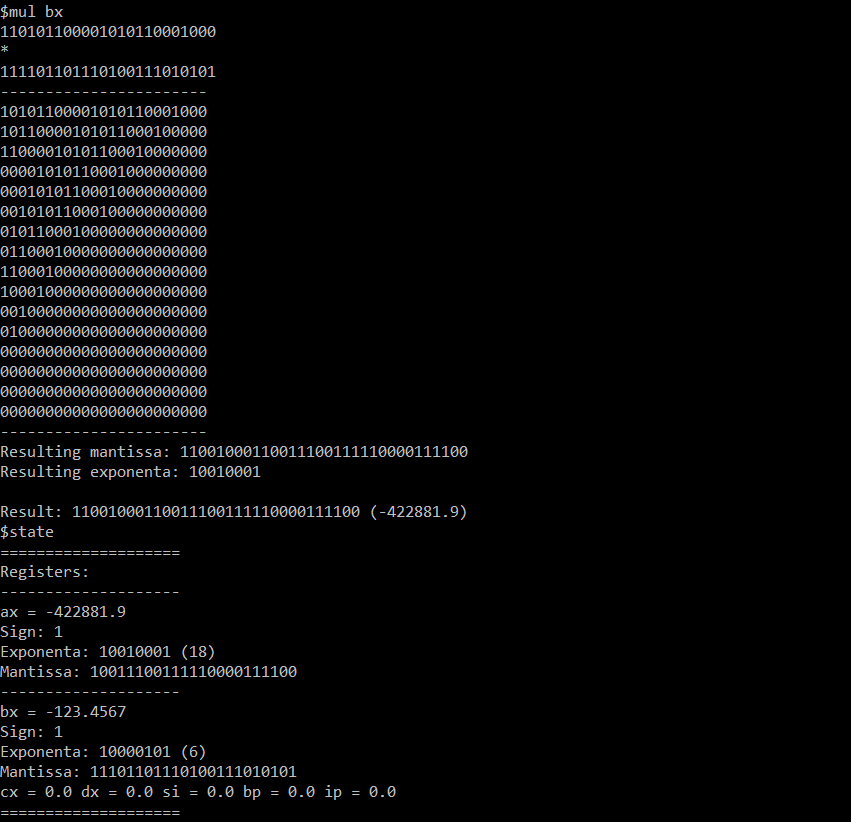


Рис4. Умножение чисел



Рис 5. Деление чисел

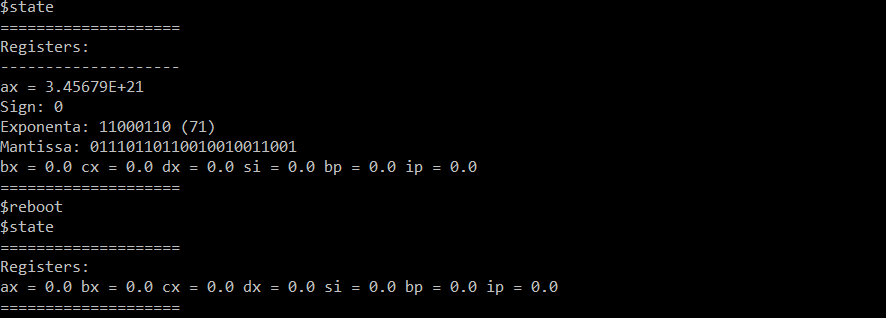


Рис 6.Просмотр состояния регистров и очистка регистров

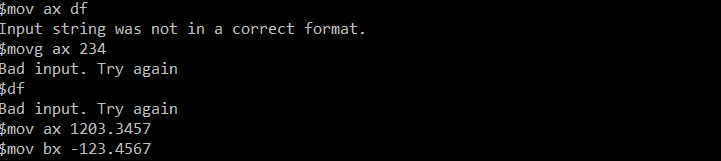


Рис 7. Неправильный ввод команды и занесение в регистр не числа

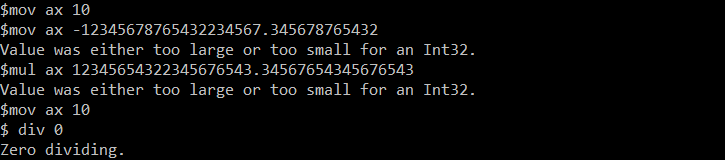


Рис 8. Обработка переполнения при добавлении слишком большого числа, обработка операции со слишком большим число, обработка деления на 0

# 6. Вывод

В процессе выполнения лабораторных работ были исследованы принципы работы процессора и АЛУ, виды представления вещественных чисел в процессоре, а также алгоритмы, по которым производятся операции с целыми числами (сложение, вычитание, умножение, деление). Было изучено представление чисел с плавающей точкой. На основе полученных знаний на языке C# была написана программа эмулятора АЛУ, пошагово выполняющего операции с вещественными числами.

# Приложение. Исходный код программы

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace oop\_4\_6

{

class Program

{

static void Main(string[] args)

{

string command;

Console.WriteLine("\t\t\t\t\tALU Emulator, Roman Martyanov");

Processor a = new Processor();

while (true)

{

Console.Write("$");

command = Console.ReadLine();

try

{

Instruction instruction = Interpreter.Parse(command);

a.Run(instruction);

}

catch (Exception e)

{

Console.WriteLine(e.Message);

}

}

}

}

}

//-----------------------------------------------------------------------------------------------

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace oop\_4\_6

{

public class Instruction

{

public string Name { get; set; }

public string Arg1 { get; set; }

public string Arg2 { get; set; }

public Instruction()

{

}

public Instruction(string name)

{

Name = name;

}

}

}

//------------------------------------------------------------------------------------------------------

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace oop\_4\_6

{

static class Interpreter

{

private static Table assemblyTable = new Table();

public static Instruction Parse(string command)

{

string[] args = System.Text.RegularExpressions.Regex.Split(command, @"\s{1,}"); ;

if (args.Length > 3)

{

throw new Exception("Wrong command");

}

Instruction instruction = new Instruction()

{

Name = args[0],

};

if (args.Length == 1)

{

return instruction;

}

instruction.Arg1 = args[1];

if (args.Length == 2)

{

return instruction;

}

instruction.Arg2 = args[2];

return instruction;

}

private static string Prepare(string command)

{

command = command.Trim().ToLower().Replace(',', ' ');

return command;

}

public static string FloatToBinary(float number)

{

int dec = BitConverter.ToInt32(BitConverter.GetBytes(number), 0);

return Convert.ToString(dec, 2);

}

public static string Normalize(string bin, int len)

{

if (bin.Length > len)

{

bin = bin.Substring(bin.Length - len, len);

}

int d = len - bin.Length;

for (int i = 0; i < d; i++)

{

bin = "0" + bin;

}

return bin;

}

}

}

//-------------------------------------------------------------------------------------------------------

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace oop\_4\_6

{

public class Processor

{

private static Dictionary<string, Register> registers;

private static ALU alu;

public static Dictionary<string, Register> Registers { get => registers; }

public Processor()

{

registers = new Dictionary<string, Register>();

alu = new ALU();

registers["ax"] = new Register(0);

registers["bx"] = new Register(0);

registers["cx"] = new Register(0);

registers["dx"] = new Register(0);

registers["si"] = new Register(0);

registers["bp"] = new Register(0);

registers["ip"] = new Register(0);

}

public void Run(Instruction instruction)

{

string name = instruction.Name;

switch (name)

{

case "stop":

Exit();

break;

case "reboot":

Reboot();

break;

case "state":

PrintState();

break;

case "mov":

Mov(instruction.Arg1, instruction.Arg2);

break;

case "add":

Add(instruction.Arg1, instruction.Arg2);

break;

case "mul":

Mul(instruction.Arg1);

break;

case "sub":

Sub(instruction.Arg1, instruction.Arg2);

break;

case "div":

Div(instruction.Arg1);

break;

default:

Console.WriteLine("Bad input. Try again");

break;

}

}

public static void Mov(string dest, string source)

{

if (!IsRegister(dest))

{

Exit();

throw new Exception("Wrong operand in mov command");

}

Register destReg = registers[dest];

if (IsRegister(source))

{

Register sourceReg = registers[source];

destReg.Val = sourceReg.Val;

destReg.Sign = sourceReg.Sign;

destReg.Exponenta = sourceReg.Exponenta;

destReg.Mantissa = sourceReg.Mantissa;

registers[dest] = destReg;

}

else

{

destReg.Init(float.Parse(source));

registers[dest] = destReg;

}

}

public static void Add(string dest, string source)

{

string op1 = "", op2 = "";

if (!IsRegister(dest))

{

throw new Exception("Bad register name");

}

op1 = registers[dest].ToString();

if (IsRegister(source))

{

op2 = registers[source].ToString();

float res = registers["ax"].Val + registers[source].Val;

if (float.IsInfinity(res)) throw new Exception("adding overflow");

registers[dest] = new Register(registers[dest].Val + registers[source].Val);

}

else

{

op2 = new Register(float.Parse(source)).ToString();

float res = registers["ax"].Val + float.Parse(source);

if (float.IsInfinity(res)) throw new Exception("adding overflow");

registers[dest] = new Register(registers[dest].Val + float.Parse(source));

}

alu.Add(op1, op2);

Console.WriteLine("\nResult: " + registers[dest] + " (" + registers[dest].Val + ")");

}

public static void Sub(string dest, string source)

{

string op1 = "", op2 = "";

if (!IsRegister(dest))

{

throw new Exception("Bad register name");

}

op1 = registers[dest].ToString();

if (IsRegister(source))

{

op2 = registers[source].ToString();

registers[dest] = new Register(registers[dest].Val - registers[source].Val);

}

else

{

op2 = new Register(float.Parse(source)).ToString();

registers[dest] = new Register(registers[dest].Val - float.Parse(source));

}

alu.Sub(op1, op2);

Console.WriteLine("\nResult: " + registers[dest] + " (" + registers[dest].Val + ")");

}

public static void Mul(string source)

{

string op1 = "", op2 = "";

op1 = registers["ax"].ToString();

if (IsRegister(source))

{

op2 = registers[source].ToString();

float res = registers["ax"].Val \* registers[source].Val;

if (float.IsInfinity(res)) throw new Exception("multiplication overflow");

registers["ax"] = new Register(registers["ax"].Val \* registers[source].Val);

}

else

{

op2 = new Register(float.Parse(source)).ToString();

Console.WriteLine("operand1: " + registers["ax"].Val);

float res = registers["ax"].Val \* float.Parse(source);

if (float.IsInfinity(res)) throw new Exception("multiplication overflow");

registers["ax"] = new Register(registers["ax"].Val \* float.Parse(source));

Console.WriteLine("operand2: " + float.Parse(source));

}

alu.Mul(op1, op2);

Console.WriteLine("\nResult: " + registers["ax"] + " (" + registers["ax"].Val + ")");

}

public static void Div(string source)

{

string op1 = "", op2 = "";

op1 = registers["ax"].ToString();

if (IsRegister(source))

{

op2 = registers[source].ToString();

if (registers[source].Val == 0) throw new Exception("Cannot divide by zero");

registers["ax"] = new Register(registers["ax"].Val / registers[source].Val);

}

else

{

op2 = new Register(float.Parse(source)).ToString();

registers["ax"] = new Register(registers["ax"].Val / float.Parse(source));

}

alu.Div(op1, op2);

Console.WriteLine("\nResult: " + registers["ax"] + " (" + registers["ax"].Val + ")");

}

public static bool IsRegister(string val)

{

foreach (var r in registers)

{

if (r.Key == val)

{

return true;

}

}

return false;

}

public static void PrintState()

{

Console.WriteLine(new string('=', 20));

Console.WriteLine("Registers: ");

foreach (var reg in registers)

{

if (reg.Value.Val != 0)

{

Console.WriteLine(new string('-', 20));

Console.WriteLine(reg.Key + " = " + reg.Value.Val);

Console.WriteLine("Sign: " + reg.Value.Sign);// + " (" +

Console.WriteLine("Exponenta: " + reg.Value.Exponenta + " (" +

reg.Value.GetExp().ToString() + ")");

Console.WriteLine("Mantissa: " + reg.Value.Mantissa);

}

else

{

Console.Write(reg.Key + " = 0.0" + " ");

}

}

Console.WriteLine();

Console.WriteLine(new string('=', 20));

}

public static void Reboot()

{

registers["ax"] = new Register(0);

registers["bx"] = new Register(0);

registers["cx"] = new Register(0);

registers["dx"] = new Register(0);

registers["si"] = new Register(0);

registers["bp"] = new Register(0);

registers["ip"] = new Register(0);

}

public static void Exit()

{

registers["ax"] = new Register(0);

registers["bx"] = new Register(0);

registers["cx"] = new Register(0);

registers["dx"] = new Register(0);

registers["si"] = new Register(0);

registers["bp"] = new Register(0);

registers["ip"] = new Register(0);

}

}

}

//--------------------------------------------------------------------------------------------------

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace oop\_4\_6

{

public class Register

{

public float Val { get; set; }

public byte Id { get; set; }

public char Sign { get; set; }

public string Mantissa { get; set; }

public string Exponenta { get; set; }

public Register(float val)

{

Init(val);

Id = 0x00;

}

private void Parse(string s)

{

Sign = s[0];

Exponenta = s.Substring(1, 8);

Mantissa = s.Substring(9, 23);

}

public int GetExp() =>

(Convert.ToInt16(Exponenta, 2) == 0 ? 127 : Convert.ToInt16(Exponenta, 2)) - 127;

public static int GetExp(string s) =>

(Convert.ToInt16(s.Substring(1, 8), 2) == 0 ? 127 : Convert.ToInt16(s.Substring(1, 8), 2)) - 127;

public override string ToString()

{

return Sign + Exponenta + Mantissa;

}

public string Print()

{

return Sign + "|" + Exponenta + "|" + Mantissa;

}

public void Init(float val)

{

Val = val;

Sign = val >= 0 ? '0' : '1';

string s = Sign + Interpreter.Normalize(Interpreter.FloatToBinary(Math.Abs(val)), 31);

Parse(s);

}

public static implicit operator Register(string s)

{

Register r = new Register(0);

r.Sign = s[0];

r.Exponenta = s.Substring(1, 8);

r.Mantissa = s.Substring(9, 23);

return r;

}

}

}

//--------------------------------------------------------------------------------------------------------------------

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace oop\_4\_6

{

public class Table

{

private Dictionary<string, byte> registers;

public Table()

{

registers = new Dictionary<string, byte>();

registers["ax"] = 0x00;

registers["bx"] = 0x01;

registers["cx"] = 0x02;

registers["dx"] = 0x03;

registers["si"] = 0x04;

registers["bp"] = 0x05;

registers["ip"] = 0x06;

}

}

}