Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Кафедра информатики

Отчет по лабораторной работе №2

Эмуляция АЛУ. Операция умножения целых чисел

Студент: гр. 853503

Осипчик Тимофей Валентинович

Руководитель: старший преподаватель

Шиманский В.В.

Минск 2020

СОДЕРЖАНИЕ

1. Введение
2. Постановка задачи
3. Программная реализация
4. Выводы

Литература

Приложение 1 – Текст программы

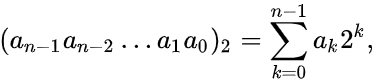
**1.Введение**

**1.1. Представление целых чисел.**

В двоичной системе счисления числа записываются с помощью двух [символов](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BC%D0%B2%D0%BE%D0%BB) (0 и 1). Чтобы не путать, в какой системе счисления записано число, его снабжают указателем справа внизу. Например, число в десятичной системе 510, в двоичной 1012. Иногда двоичное число обозначают префиксом 0b или символом & (амперсанд), например, 0b101 или соответственно &101.

В двоичной системе счисления (как и в других системах счисления, кроме десятичной) знаки читаются по одному. Например, число 1012 произносится «один ноль один».

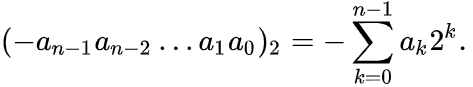
Натуральное число, записываемое в двоичной системе счисления как  {\displaystyle (a\_{n-1}a\_{n-2}\dots a\_{1}a\_{0})\_{2}}(*an – 1 an – 2 ... a1 a0*)2, имеет значение:

{\displaystyle (a\_{n-1}a\_{n-2}\dots a\_{1}a\_{0})\_{2}=\sum \_{k=0}^{n-1}a\_{k}2^{k},}

где:

* {\displaystyle n}*n* — количество [цифр](https://ru.wikipedia.org/wiki/%D0%A6%D0%B8%D1%84%D1%80%D1%8B) (знаков) в числе,
* {\displaystyle a\_{k}}*ak* — цифры из множества {0,1},
* {\displaystyle k}*k* — порядковый номер цифры.

Отрицательные двоичные числа обозначаются так же, как и десятичные: знаком «−» перед числом. А именно, отрицательное целое число, записываемое в двоичной системе счисления {\displaystyle (-a\_{n-1}a\_{n-2}\dots a\_{1}a\_{0})\_{2}}(-*an – 1 an – 2 ... a1 a0*)2, имеет величину:



В вычислительной технике широко используется запись отрицательных двоичных чисел в дополнительном коде.

**1.2. Прямой код.**

При записи числа в **прямом коде** (англ. *Signed magnitude representation*) старший разряд является знаковым разрядом. Если его значение равно нулю, то представлено положительное число или положительный ноль, если единице, то представлено отрицательное число или отрицательный ноль. В остальных разрядах (которые называются цифровыми) записывается двоичное представление модуля числа. Например, число −5 в восьмибитном типе данных, использующем прямой код, будет выглядеть так: 10000101.

Таким способом в n-битовом типе данных можно представить диапазон чисел [−2n – 1 + 1;2n – 1 −1].

**Достоинства представления чисел с помощью прямого кода**

1. Получить прямой код числа достаточно просто.
2. Из-за того, что 0 обозначает +, коды положительных чисел относительно беззнакового кодирования остаются неизменными.
3. Количество положительных чисел равно количеству отрицательных.

**Недостатки представления чисел с помощью прямого кода**

1. Выполнение арифметических операций с отрицательными числами требует усложнения архитектуры центрального процессора (например, для вычитания невозможно использовать сумматор, необходима отдельная схема для этого).
2. Существуют два нуля: −0 (100…000) и +0 (000…000), из-за чего усложняется арифметическое сравнение.

Из-за весьма существенных недостатков прямой код используется очень редко.

**1.3. Дополнительный код.**

В качестве альтернативы представления целых чисел может использоваться код с **дополнением до единицы** (англ. *Ones' complement*).

Алгоритм получения кода числа:

* если число положительное, то в старший разряд (который является знаковым) записывается ноль, а далее записывается само число;
* если число отрицательное, то код получается инвертированием представления модуля числа (получается обратный код);
* если число является нулем, то его можно представить двумя способами: +0 (000…000) или −0 (111…111).

Пример: переведём число −13−13 в двоичный восьмибитный код. Прямой код модуля −13: 00001101, инвертируем и получаем 11110010. Для получения из дополнительного кода самого числа достаточно инвертировать все разряды кода.

Таким способом можно получить диапазон значений [−2n – 1 +1;2n – 1 −1].

**Достоинства представления чисел с помощью кода с дополнением до единицы**

1. Простое получение кода отрицательных чисел.
2. Из-за того, что 0 обозначает +, коды положительных чисел относительно беззнакового кодирования остаются неизменными.
3. Количество положительных чисел равно количеству отрицательных.

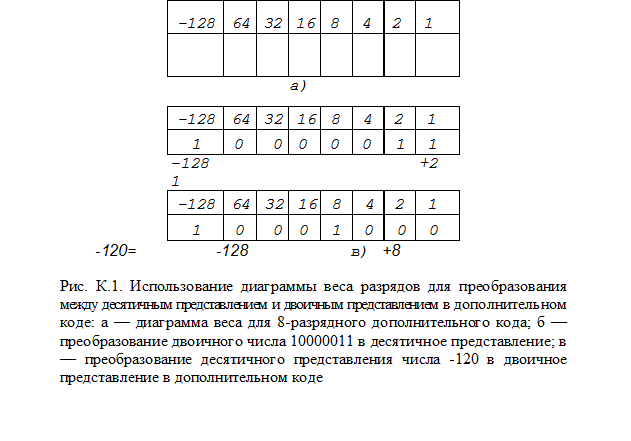
**Недостатки представления чисел с помощью кода с дополнением до единицы**

1. Выполнение арифметических операций с отрицательными числами требует усложнения архитектуры центрального процессора.
2. Существуют два нуля: +0 и −0.

|  |  |  |
| --- | --- | --- |
| Десятичное  представление | Прямой код | Дополнительный код |
| +127 | *0111 1111* | *0111 1111* |
| +1 | *0000 0001* | *0000 0001* |
| +0 | *0000 0000* | *0000 0000* |
| -0 | *1000 0000* | *0000 0000* |
| -1 | *1000 0001* | *1111 1111* |
| -2 | *1000 0010* | *1111 1110* |
| -3 | *1000 0011* | *1111 1101* |
| -4 | *1000 0100* | *1111 1100* |
| -5 | *1000 0101* | *1111 1011* |
| -6 | *1000 0110* | *1111 1010* |
| -7 | *1000 0111* | *1111 1001* |
| *-8* | *1000 1000* | *1111 1000* |
| *-9* | *1000 1001* | *1111 0111* |
| *-10* | *1000 1010* | *1111 0110* |
| *-11* | *1000 1011* | *1111 0101* |
| *-127* | *1111 1111* | *1000 0001* |
| *-128* | *-* | *1000 0000* |

Таблица 1. Варианты двоичного 8-разрядного представления целых чисел

Хорошей иллюстрацией принципа представления в дополнительном коде является диаграмма веса разрядов (рис. 1), в которой показано, что вес самого младшего разряда (крайней правой позиции на диаграмме) равен 1 (т.е. 20). Вес каждого последующего — возрастает вдвое, и так до крайней левой позиции, знак веса которой инвертируется. Рис. 1-а дает представление о том, почему максимальное по абсолютной величине отрицательное число, которое можно представить в дополнительном коде, равно -2n-1. Код 1 в любом значащем разряде означает добавление во взвешенную сумму положительного числа, равного весу этого разряда. Очевидно также, что положительные числа должны иметь в знаковом разряде код 0, а отрицательные — код 1. Следовательно, самое большое положительное число должно иметь в знаковом разряде код 0, а во всех значащих — код 1 и будет равно 2n-1-1.



На рис. К.1 также показано, как можно использовать диаграмму веса разрядов для преобразования из десятичного представления в двоичное и наоборот.

1.4. Сложение и вычитание в дополнительном коде.

Рассмотрим примеры:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *(а)* | *-7=* | *1001* |  | *(б)* | *-4=* | *1100* |  |
|  | *+5=* | *+0101* |  |  | *+4=* | *+0100* |  |
|  |  | *1110* | *=-2* |  |  | ***1****0000* | *=0* |
| *(в)* | *+3=* | *0011* |  | *(г)* | *-4=* | *1100* |  |
|  | *+4-* | *+0100* |  |  | *-1=* | *+1111* |  |
|  |  | *0111* | *=7* |  |  | *11011* | *=-5* |
| *(д)* | *+5=* | *0101* |  | *(е)* | *-7=* | *1001* |  |
|  | *+4=* | *+0100* |  |  | *-6=* | *+1010* |  |
|  |  | *1001* | *переполнение* |  |  | ***1****0011* | *переполнение* |

Первые четыре примера демонстрируют успешное выполнение операций. Если результат операции должен быть положительным, получается код положительного числа в дополнительном коде, а если отрицательным — код отрицательного числа в дополнительном коде. Обратите внимание на то, что в примере (г) формируется перенос из старшего (знакового) разряда, который игнорируется.

При выполнении сложения чисел с одинаковыми знаками результат может оказаться таким, что не вмещается в используемую разрядную сетку, т.е. получается число, которое выходит за диапазон представления. Появление такого результата расценивается как переполнение (overflow), и на схему АЛУ возлагается функция выявить переполнение и выработать сигнал, который должен воспрепятствовать использованию в дальнейшем полученного ошибочного результата. Существует следующее правило обнаружения переполнения:

Если знаки слагаемых совпадают, то переполнение возникает в том и только в том случае, когда знак суммы, полученной по правилам сложения в дополнительном коде, отличается от знака слагаемых.

Примеры (д) и (е) иллюстрируют появление переполнения при сложении положительных и отрицательных чисел. Обратите внимание на то, что переполнение может появиться и в том случае, когда возникает перенос из знакового разряда и когда перенос не возникает.

Операция вычитания выполняется по следующему правилу: Для вычитания одного числа (вычитаемого) из другого (уменьшаемого) необходимо предварительно выполнить операцию отрицания над вычитаемым, а затем сложить результат с уменьшаемым по правилам сложен дополнительном коде.

Примеры выполнения с различными знаками.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *(а)   М= 2 =* | *0010* | | | | *(б)* *М= 5 =               0101* | | | |
| *S= 7 =* | *0111* | | | | *S= 2 =               0010* | | | |
| *-S =* | *1001* | | | | *-S =               1110* | | | |
|  | *0010* | | | | *0101* | | | |
|  | *+ 1001* | | | | *+  1110* | | | |
|  | *1011* | | | *=-5* | *3=* | | ***1****0011* | |
| *(в)   М=-5 =* | *1011* | | | | *(г)* *М= 5 =               0101* | | | |
| *S=2 =* | *0010* | | | | *S=-2 =               1110* | | | |
| -S = | *1110* | | | | *-S =                  0010* | | | |
|  | *1011* | | | | *0101* | | | |
|  | *+ 1110* | | | | *+  0010* | | | |
|  | ***1****1001* | | *= -7* | | *7=* | | | *0111* |
| *(д)   М= 7 =* | *0111* | | | | *(е)*  *М=-6 =              1010* | | | |
| *S=-7 =* | *1001* | | | | *S= 4 =              0100* | | | |
| *-S =* | *0111* | | | | *-S=                 1100* | | | |
|  | *0111* | | | | *1010* | | | |
|  | *+ 0111* | | | | *+  1100* | | | |
|  | *1110* | *переполнение* | | | *переполнение* | ***1****0110* | | |
|  |  |  |  |  |  |  |  |  |

На рис. К.2 представлена блок-схема узлов АЛУ, принимавших участие в выполнении операций сложения и вычитания целых чисел. Центральным узлом является двоичный сумматор, на входы которого подают коды слагаемых, а на выходах формируется двоичный код суммы, причём реализация выполняется по правилам сложения чисел без знака. При выполнении сложения оба слагаемых направляются на входы сумматора непосредственно из регистров слагаемых А и В.

Результат передается либо в один из регистров слагаемых (этот вариант показан на схеме), либо в третий регистр результата. Кроме кода результата сумматор формирует сигнал переполнения, который фиксируется в битовом флаге переполнения. Значение флага интерпретируется следующим образом: 0 — переполнение отсутствует, 1 – присутствует. При выполнении операции вычитания код вычитаемого, хранящийся перед началом операции в регистре В, передается на схему, выполняющую операцию отрицания, а уже с выхода этой схемы код поступает на вход сумматора.

1.4. произведение в дополнительном коде.

При точном умножении двух чисел количество значащих цифр произведения превышает количество значащих цифр сомножителей, в пределе доходя до двойного количества значащих цифр. При умножении нескольких чисел количество значащих цифр может оказаться ещё больше. Ограниченное количество разрядов в устройствах цифровой машины вынуждает, как правило, ограничиваться в произведении тем же количеством значащих цифр, которые имели сомножители.

При умножении сомножителей, имеющих только дробную часть, возможен выход за пределы разрядной сетки только со стороны младших разрядов. Ограничение количества значащих разрядов этого произведения вносит в произведение погрешность отрицательного знака, т.е. произведение вычисляется с недостатком.

При большом объеме вычислений ошибки одного знака накладываются друг на друга, в результате чего общая ошибка сильно возрастает. Поэтому существенное значение с точки зрения накопления ошибок при большом объеме вычислений имеет округление результатов умножения, из-за которого ошибка произведения должна стать знакопеременной, а математическое ожидание ошибки при условии, что отброшенные младшие разряды могут с одинаковой вероятностью иметь любое из возможных значений, становится равным нулю. При этом предельное по абсолютной величине значение ошибки становится наименьшим из возможных при заданном количестве значащих цифр, т.е. равным половине младшего разряда. Умножение двух сомножителей е произвольным сочетанием знаков удобно выполнять, если сомножители заданы в прямом коде. В этом случае, независимо от знаков, модуль произведения определяется обычным способом, а знак произведения определяется как сумма знаков обоих сомножителей по модулю 2. Если оба сомножителя имеют одинаковые знаки, то сумма знаков по модулю 2 будет равна 0, что означает, что произведение положительно. Если оба сомножителя имеют разные знаки, то сумма знаков равна 1, что означает, что произведение отрицательно.

Для выполнения операции умножения наиболее удобным является прямой код, поскольку при этом умножение сводится к двум независимым операциям, которые выполняются одновременно: перемножению цифровых частей чисел и суммированию цифр знаковых разрядов по модулю 2.

Так как в прямом коде цифровые части чисел соответствуют абсолютным значениям их, то получающийся результат умножения точно соответствует произведению исходных чисел, и никаких поправок результата, кроме присвоения ему знака, не требуется.

Однако прямой код неудобен для выполнения операций сложения (вычитания).

Таким образом, если в машине числа представлены инверсным кодом, то операции умножения в прямом коде должна предшествовать проверка знаков сомножителей и преобразование отрицательных сомножителей из дополнительного или обратного кода в прямой. Кроме того, если произведение отрицательно, го по окончании операции его следует из прямого кода преобразовать в дополнительный или обратный. Эти дополнительные такты операции умножения удлиняют и усложняют её и усложняют управление ею. Правда, затраты времени на эти дополнительные такты при умножении относительно не так велики, поэтому умножение в прямом коде встречается очень часто.

При отрицательных сомножителях возможно умножение не самих чисел, а их дополнительных или обратных кодов. Как и при сложении (вычитании) в этом случае требуется введение поправок для получения правильного результата умножения.

Методы введения поправок при умножении получаются сложнее, чем при сложении. Содержание их зависит, во-первых, от того, какой код - дополнительный или обратный - принят для записи отрицательных сомножителей, и вовторых, от того, как распределены сдвиги между регистрами множимого, множителя и частичных произведений, то есть по какому из вариантов осуществляется умножение.

Поскольку имеется четыре варианта распределения сдвигов (рассмотренные выше четыре варианта выполнения умножения) и два способа записи отрицательных чисел - дополнительный и обратный коды, - очевидно, возможно восемь различных схем умножения, которые предусматривают введение поправок - 43 - в произведение.

Сложность методов коррекции при перемножении чисел, записанных дополнительным или обратным кодом, обусловлено тем, что в исправлении нуждается не только знак, но и цифровая часть произведения.

Действительно, непосредственное перемножение кодов отрицательных чисел не дает нужного результата.

Пример:  
  
Выполнить умножение чисел **A = 1210** и **B = 910** в двоичной системе счисления.

**Решение:**  
  
**1)** Переведем числа в двоичную систему счисления:  
    1210 = **1100**2  
    910 = **1001**2  
  
**2)** Запишем числа "A" и "B" столбиком, одно под другим, начиная с младших разрядов (нумерация разрядов начинается с нуля).

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Разр. | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| A |  |  |  |  |  | **1** | **1** | **0** | **0** |
| B |  |  |  |  |  | **1** | **0** | **0** | **1** |

**3) Умножим** последовательно все разряды числа "A" на младший разряд "B" записывая результат внизу, под сомножителями, начиная с младших разрядов.

**Правила поразрядного умножения**, для двоичной системы счисления, очень просты. В результирующем разряде будет единица только в одном случае, когда оба сомножителя содержат единицы в соответсвующем разряде. Во всех других случаях в результирующем разряде будет ноль. Это можно записать в виде простой таблицы:

|  |  |  |
| --- | --- | --- |
| Ai | Bi | Ci |
| **0** | **0** | **0** |
| **0** | **1** | **0** |
| **1** | **0** | **0** |
| **1** | **1** | **1** |

**4)** Таким же образом умножим последовательно все разряды числа "A" на следующий разряд "B" записывая результат внизу, под сомножителями, со сдвигом на один разряд влево.

**5)** Сложим полученные на последних двух шагах двоичные числа. Как это сделать можно посмотреть в примере на сложение.

Шаги 4 и 5 будем повторять до тех пор, пока не исчерпаем все разряды множителя.  
Весь процесс умножения наших чисел выглядит следующим образом:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Разр. | c | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|  |  |  |  |  |  |  |  |  |  |  |
| A |  |  |  |  |  |  | **1** | **1** | **0** | **0** |
| B |  |  |  |  |  |  | **1** | **0** | **0** | **1** |
|  |  |  | | | | | | | | |
| 1100 x 1 |  |  |  |  |  |  | **1** | **1** | **0** | **0** |
| 1100 x 0 |  |  |  |  |  |  |  |  | **0** |  |
|  |  |  | | | | | | | | |
| 1100 + 0 |  |  |  |  |  |  | **1** | **1** | **0** | **0** |
| 1100 x 0 |  |  |  |  |  |  |  | **0** |  |  |
|  |  |  | | | | | | | | |
| 1100 + 0 |  |  |  |  |  |  | **1** | **1** | **0** | **0** |
| 1100 x 1 |  |  |  | **1** | **1** | **0** | **0** |  |  |  |
|  |  |  | | | | | | | | |
| 1100 + 1100000 |  |  |  | **1** | **1** | **0** | **1** | **1** | **0** | **0** |

**Ответ:**    **1100**2 x **1001**2 = **1101100**2  
  
или в десятичной системе счисления: 1210 x 910 = 10810

2. Постановка задачи

2.1. Текст задания

Эмуляция АЛУ. Реализовать операции сложения и вычитания целых чисел. Использовать дополнительный код.

2.2. Примечание к заданию

Реализовать ввод двух чисел. Вычислить сумму и разность в двоичной системе счисления в дополнительном коде и вывести результат на экран.

3. Программная реализация

3.1. С консоли вводятся два числа в десятичной системе счисления. Затем реализуется перевод их в двоичную систему счисления. После двоичный код преобразуется в дополнительный. В дополнительном коде реализуется сложение и вычитание. Затем на экран выводится результаты сложения и вычитания в двоичной и десятичной системах счисления.

3.2. Примеры

3.2.1. Тест для «a = 10, b = 4»

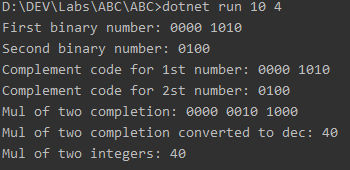


Рисунок 1. Скриншот для «a = 10, b = 4»

Ответ: Произведение = 40

3.2.2. Тест для «a = 9, b = -5»

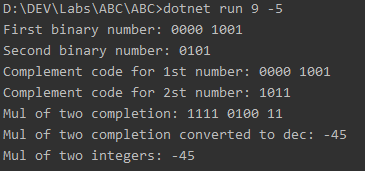


Рисунок 2. Скриншот для «a = 9, b = -5»

Ответ: Произведение = -45.

3.2.3. Тест для «a = -154, b = 100»

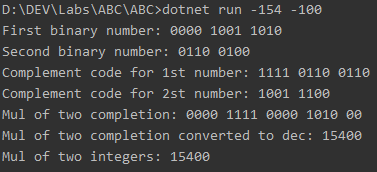


Рисунок 3. Скриншот для «a = -154, b = 100»

Ответ: Произведение = -15400.

3.2.4. Тест для «a = -1432, b = -104»

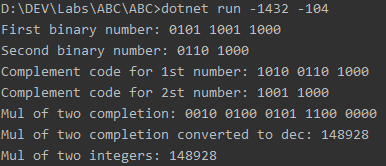


Рисунок 4. Скриншот для «a = -1432, b = -104»

Ответ: Произведение = 148928.

3.2.5. Тест для «a = 6666666666666666666, b = 3333333333333333333»

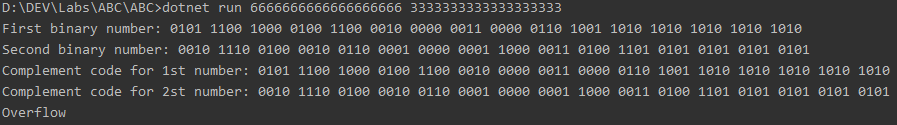


Рисунок 5. Скриншот для «a = 6666666666666666666, b = 3333333333333333333»

Ответ: Переполнение разрядной сетки.

3.2.6. Тест для «a =2222222222222222222, b = 2»

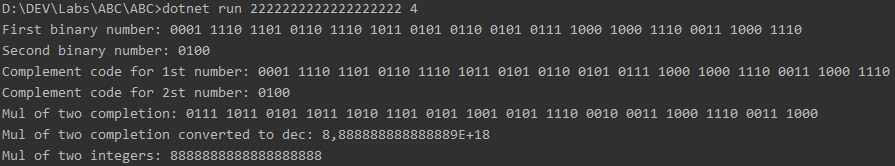


Рисунок 6. Скриншот для «a =2222222222222222222, b = 2»

Ответ: Произведение = 8888888888888888888.

3.2.6. Тест для «a =0, b = 0»

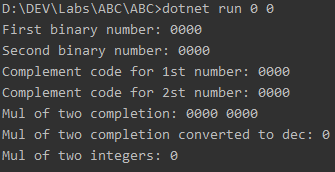


Рисунок 6. Скриншот для «a =0, b = 0»

Ответ: Произведение = 0.

3.2.7. Резюме.

В своих примерах я старался рассмотреть максимально разнообразный набор случаев. Были рассмотрены произведение двух положительных чисел, двух отрицательных чисел, отрицательного и положительного чисел.

Также были рассмотрены экстремальные случаи поведения программы: длинные числа, неправильный ввод.

4. Выводы

В ходе выполнения лабораторной работы ознакомился с понятиями прямого кода, дополнительного кода, сложения и вычитания в прямом и дополнительном коде, изучил различные алгоритмы сложения и вычитания, и произведения бинарных чисел в дополнительном коде.

Так же рассмотрел работу арифметико-логического устройства (АЛУ), рассмотрел различные нюансы использования АЛУ. Для реализации использовал язык С# и интегрированную среду разработки Rider в ОС Windows 10.

Литература

1. Волорова Н. А. Лабораторный практикум по курсу «Архитектура вычислительных систем» для студентов специальности «Информатика» /985-444-487-2 – Мн.: БГУИР, 2003. — 32 с.: ил.

2. Рихтер, Дж. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C# / Дж. Рихтер. – 4-е изд. – Спб.: Питер, 2013. – 896 с.: ил.

3. Rider, ide, сайт. –URL: <https://www.jetbrains.com/rider/>

ПРИЛОЖЕНИЕ 1

Текст программы

Модуль 1. Program.cs

using System;  
using System.Collections.Generic;   
using System.Linq;  
using System.Text.RegularExpressions;   
  
namespace ABC  
{  
 public class ConsoleMain  
 {  
 private enum Operations  
 {  
 **Sum**, **Sub**, **Mul**, Div  
 }  
   
 private static void Main(string[] args)  
 {  
 if (!ReadArgs(args, out var numbers))  
 {  
 Console.WriteLine($"Incorrect input: {string.Join(", ", numbers)} it must be 2 integers");   
 return;  
 }  
 Console.WriteLine($"First binary number: {new Binary(Math.Abs(numbers[0]))}");  
 Console.WriteLine($"Second binary number: {new Binary(Math.Abs(numbers[1]))}");  
 Console.WriteLine($"Complement code for 1st number: {new Binary(numbers[0])}");  
 Console.WriteLine($"Complement code for 2st number: {new Binary(numbers[1])}");  
 try  
 {  
 Console.WriteLine($"Mul of two completion: {Calculate(numbers, Operations.**Mul**)}");  
 Console.WriteLine($"Mul of two decimals: {numbers[0] \* numbers[1]}");  
 }  
 catch (OverflowException e)  
 {  
 Console.WriteLine(e.Message);  
 }  
 }  
   
 private static bool ReadArgs(IEnumerable<string> args, out decimal[] numbers)  
 {  
 var number = new List<decimal>();  
 foreach (var s in args)  
 {  
 if (decimal.TryParse(s, out var num))  
 {  
 number.Add(num);  
 }  
 }  
   
 numbers = number.ToArray();  
 return numbers.Length == 2;  
 }  
   
 private static Binary Calculate(IReadOnlyList<decimal> numbers, Operations operation)  
 {  
 return operation switch  
 {  
 Operations.**Sum** => new Binary(numbers[0]) + new Binary(numbers[1]),  
 Operations.**Sub** => new Binary(numbers[0]) + new Binary(numbers[1] \* -1),  
 Operations.**Mul** => new Binary(numbers[0]) \* new Binary(numbers[1]),  
 \_ => throw new ArgumentException($"unknown operation: {operation}")  
 };  
 }  
 }  
}

private enum BinarySystem  
 {  
 **Base** = 2,  
 **MinLen** = 4,  
 **MaxLength** = 64  
 }  
   
 public IEnumerable<int> Value { get; }  
  
 public Binary(decimal num)   
 {  
 Value = num < 0 ? ToDirectCode(-num) : ToDirectCode(num);  
 if (num < 0)  
 {  
 Value = ToComplementCode();  
 }  
 }  
  
 public Binary(IEnumerable<int> binary)  
 {  
 Value = binary;  
 }  
   
 private static IEnumerable<int> ToDirectCode(decimal num)  
 {  
 var binary = new List<int>();  
 while (num > 0)  
 {  
 num /= (int) BinarySystem.**Base**;  
 binary.Insert(0, Convert.ToInt32(num > Math.Truncate(num)));  
 num = Math.Truncate(num);  
 }  
  
 return NormalizeNumber(binary);  
 }  
  
 private static IEnumerable<int> NormalizeNumber(IEnumerable<int> binary)  
 {  
 var number = binary.ToList();  
 return number.Count >= (int) BinarySystem.**MaxLength** ? number   
 : number.ExpandBegin(Math.Abs(number.Count % (int) BinarySystem.**MinLen** - (int) BinarySystem.**MinLen**));  
 }  
   
 private Binary ToInvertCode()  
 {  
 var binary = new int[Value.Count()];  
 var count = 0;  
 foreach (var i in Value)  
 {  
 binary[count++] = i == 1 ? 0 : 1;  
 }  
   
 return new Binary(binary);  
 }  
  
 public IEnumerable<int> ToComplementCode()  
 {  
 var code = this;  
 if (Value.First() != 1)  
 {  
 code = ToInvertCode() + new Binary(1);  
 *//code.Value.ToList()[0] = 1;* var a = code.Value.ToList();  
 a[0] = 1;  
 code = new Binary(a);  
 }  
   
 return code.Value;  
 }  
  
 public Binary ComplementToDirect => Value.First() != 1 ? this : (this + new Binary(-1)).ToInvertCode();  
   
 public static Binary operator +(Binary binary1, Binary binary2)  
 {  
 var (item1, item2) = NormalizeLists(binary1, binary2);  
 var res = CalculateSum(item1, item2);  
  
 return new Binary(res);  
 }  
   
 public static Binary operator \*(Binary binary1, Binary binary2)  
 {  
 var (item1, item2) = NormalizeLists(binary1.ComplementToDirect, binary2.ComplementToDirect);  
 var res = new Binary(new List<int>().ExpandBegin(item1.Count));  
 var shift = 0;  
 item2.Reverse();  
   
 foreach (var i in item2)  
 {  
 if (i != 0)  
 {  
 var shifted = item1.ExpandEnd(shift);  
 res += new Binary(shifted);  
 }  
 shift++;  
 }  
  
 return NormalizeMul(binary1, binary2, new Binary(NormalizeNumber(res.Value)));  
 }  
  
 private static Binary NormalizeMul(Binary binary1, Binary binary2, Binary result)  
 {  
 if (binary1.Value.First() == binary2.Value.First())  
 {  
 result = result.ComplementToDirect;  
 }  
 else if (binary1.Value.First() != binary2.Value.First() && result.Value.First() != 1)  
 {  
 result = new Binary(result.ToComplementCode());  
 }  
  
 return result;  
 }  
  
 public static Binary operator /(Binary binary1, Binary binary2)  
 {  
 if (!SqueezeBin(binary2.Value).Any())  
 {  
 throw new DivideByZeroException("Attempted to divide by zero.");  
 }  
   
 var sign = binary1.Value.First() == binary2.Value.First();  
 var binNum = binary1.ComplementToDirect.Value.ToArray();  
  
 var divider = binary2.Value.First() != 1 ? new Binary(binary2.ToComplementCode()) : binary2;  
  
 var res = GetIntPath(binNum, divider, out \_);  
  
 var asd = sign ? NormalizeNumber(SqueezeBin(res)) : new Binary(res).ToComplementCode();  
 return new Binary(asd);  
 }  
  
 private static IEnumerable<int> GetIntPath(IEnumerable<int> dividendNum, Binary divider, out List<int> dividend)  
 {  
 var res = new List<int>();  
 dividend = new List<int>();  
 foreach (var i in dividendNum)  
 {  
 dividend.Add(i);  
 var remains = new Binary(NormalizeNumber(dividend)) + divider;  
 if (remains.Value.First() != 1)  
 {  
 dividend = remains.Value.ToList();  
 res.Add(1);  
 }  
 else  
 {  
 res.Add(0);  
 }  
 }  
  
 return res;  
 }  
  
 private static IEnumerable<int> SqueezeBin(IEnumerable<int> bin)  
 {  
 var list = bin.ToList();  
 var index = list.IndexOf(1);  
 list.RemoveRange(0, index > 0 ? index : list.Count);  
   
 return list;  
 }  
   
 private static List<int> NormalizeBinary(Binary binary, int lenSub)  
 {  
 var bin = binary.Value.ToList();  
   
 return bin.ExpandBegin(lenSub, bin[0] == 1 ? 1 : 0);  
 }  
  
 private static (List<int>, List<int>) NormalizeLists(Binary binary1, Binary binary2)  
 {  
 var bin1 = binary1.Value.ToList();  
 var bin2 = binary2.Value.ToList();  
 bin1 = bin1.Count < bin2.Count ? NormalizeBinary(binary1, bin2.Count - bin1.Count) : bin1;  
 bin2 = bin2.Count < bin1.Count ? NormalizeBinary(binary2, bin1.Count - bin2.Count) : bin2;  
  
 return (bin1, bin2);  
 }  
  
 private static List<int> CalculateSum(List<int> list1, List<int> list2)  
 {  
 var addToNext = 0;  
 for (var i = list1.Count - 1; i >= 0; i--)  
 {  
 list1[i] = Add(list1[i] += list2[i] + addToNext, out var add);  
 addToNext = add;  
 }  
  
 if (list1.Count >= (int) BinarySystem.**MaxLength** && addToNext != 0)  
 {  
 throw new OverflowException("Overflow");  
 }  
   
 return list1;  
 }  
   
 private static int Add(int binItem, out int add)  
 {  
 add = 0;  
 if (binItem >= (int) BinarySystem.**Base**)  
 {  
 binItem -= (int) BinarySystem.**Base**;  
 add = 1;  
 }  
  
 return binItem;  
 }  
  
 public double ToDouble()  
 {  
 var res = 0d;  
 var bin = (new Binary(Value)).ComplementToDirect.Value;  
 var enumerable = bin as int[] ?? bin.ToArray();  
   
 var pow = enumerable.Count() - 1;  
   
 foreach (var i in enumerable)  
 {  
 res += i \* Math.Pow(2, pow);  
 pow--;  
 }  
  
 return Value.First() == 1 ? -res : res;  
 }  
   
 public override string ToString()  
 {  
 return Regex.Replace(string.Join("", Value), "**.{4}**", "$0 ");  
 }  
}