Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Кафедра информатики

Отчет по лабораторной работе №3

Эмуляция АЛУ. Операция деления 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

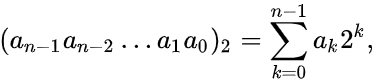
1.Введение

**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».

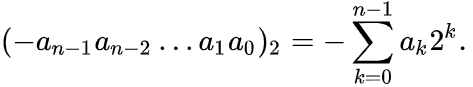
Натуральное число, записываемое в двоичной системе счисления как  {\displaystyle (a\_{n-1}a\_{n-2}\dots a\_{1}a\_{0})\_{2}}(*an – 1 an – 2 ... a1 a0*)2, имеет значение:

{\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.

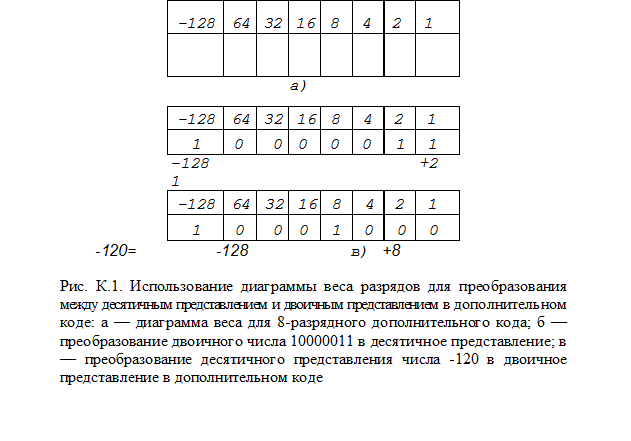
**Недостатки представления чисел с помощью кода с дополнением до единицы**

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.

**1.5. Деление в дополнительном коде.**

Рассмотрим два варианта деления целых чисел в дополнительном коде:

1) перевод операндов в прямой код, получение частного обычным способом и перевод его в дополнительный код перед записью в память;

2) деление операндов в дополнительном коде с получением частного в требуемом виде.

Второй путь так же прост, как и первый. Необходимо только управлять логикой образовании цифр частного, чему способствуют два обстоятельства:

1) делимое в процессе операции участвует всего лишь один раз, а дальше текущие остатки получаются автоматически;

2) нетрудно организовать получение цифр дополнительного кода отрицательного частного непосредственно в процессе деления путем передачи знаковых цифр остатков из СМ в регистр частного напрямую, минуя БИЦ.

Рассмотрим четыре возможных случая, определяемые комбинациями знаков дополнительных кодов операндов.

Деление производится как обычно: знак частного определяется путем сложения знаковых цифр операндов по модулю 2 и одновременно формируется дополнение модуля делителя до двух. Цифры частного получаются, как уже известно, путем инвертирования знаковых цифр остатков от деления в БИЦ.

Ход деления обычный с учетом следующих особенностей.

Псевдознаковая цифра частного (сигнал о возможном переполнении разрядной сетки) должна получаться в этом случае равной знаковой цифре первого остатка, так как здесь производится вычитание модуля делимого из модуля делителя в силу исходного сочетания знаков операндов. Значит знаковая цифра нулевого остатка должна поступать в регистр частного, минуя БИЦ, т. е. не инвертируясь.

Так как знак нулевого остатка имеет инверсное значение, то при продолжении деления обычным путем все другие остатки получаются также с инверсными знаками. Значит, в регистр частного также будут записываться инверсные цифры и в конце деления в нем образуется обратный код отрицательного частного. Для округления результата из него надо вычесть единицу первого разряда, а для перевода, его в дополнительный код к нему надо прибавить единицу во второй разряд. Тогда прибавление единицы в третий разряд произведет суммарное действие округления и перевода обратного кода в дополнительный.

.

Этот случай является зеркальным отображением предыдущего: псевдознаковая цифра частного получается как обычно путем инверсии знака первого остатка, а для того чтобы получить инверсные цифры частного, нужно брать их равными знаковым цифрам соответствующих остатков без инвертирования их в БИЦ. В конце деления необходимо добавить к четвертому разряду частного единицу для одновременного выполнения округления результата и перевода его в дополнительный код.

В этом случае все цифры частного (включая псевдознаковую) должны быть равны знаковым цифрам соответствующих остатков. На первом этапе деления одновременно с определением истинного знака частного определяется величина модуля делителя, т. е. возникает ситуация, сходная с начальными условиями 2-го случая. Значит, псевдознаковая цифра частного определяется значением знаковой цифры первого остатка. Если далее производить деление обычным путем, то, как уже известно из 2-го случая, будут получаться инверсные цифры модуля частного. Следовательно, знаковые цифры остатков надо посылать в регистр частного, минуя БИЦ, так как частное здесь должно быть положительным и его дополнительный код совпадает с прямым.

Таким образом, деление чисел, представленных в дополнительном коде, производится в 3 этапа.

На первом этапе определяется знак частного путем сложения знаковых цифр операндов по модулю 2.

На втором этапе производится нулевой шаг деления для проверки частного на переполнение разрядной сетки путем алгебраического сложения, делимого с делителем, которому приписывается знак, противоположный знаку делимого. Псевдознаковая цифра модуля частного определяется как обратное значение знаковой цифры первого остатка, если делимое положительное. Псевдознаковая цифра берется тождественно равной знаку первого остатка, если делимое отрицательное.

На третьем этапе производятся все последующие пять шагов деления по обычным правилам, но с той лишь разницей, что при отрицательном делителе цифры частного берутся тождественно равными знаковым цифрам соответствующих остатков. В конце деления в шестой разряд частного обязательно добавляется 1.

2. Постановка задачи

2.1. Текст задания

Эмуляция АЛУ. Реализовать операцию деления целых чисел.

2.2. Примечание к заданию

Реализовать ввод двух чисел. Вычислить частное 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 деление. Затем на экран выводятся результаты произведения в двоичной и десятичной системах счисления.

3.2. Примеры

3.2.1. Tест для «a = 0, b = 0»

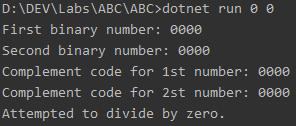


Рисунок 4. Скриншот для «a = 0, b = 0»

Ответ: попытка делить на ноль.

3.2.2. Тест для «a = 23, b = 0»

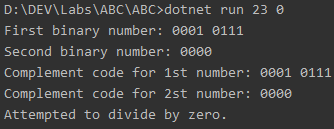


Рисунок 5. Скриншот для «a = 23, b = 0»

Ответ: попытка делить на ноль.

3.2.4. Тест для «a = 20, b = 1000»

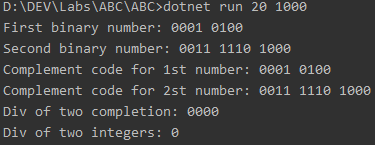


Рисунок 4. Скриншот для «a = 20, b = 1000»

Ответ: Результат деления = 0.

3.2.5. Тест для «a = 25, b = 5»

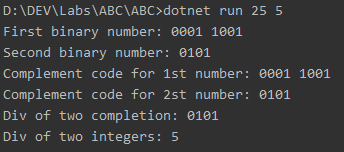


Рисунок 4. Скриншот для «a = 25, b = 5»

Ответ: Результат деления = 5.

3.2.6. Тест для «a = 148148148147851851851852, b = 444444444444»

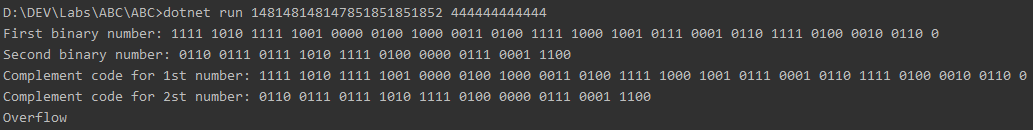


Рисунок 4. Скриншот для «a = 148148148147851851851852, b = 444444444444»

Ответ: переполнение разрядной сетки.

3.2.7. Тест для «a = 1481481481478518518, b = 44444444444444444»

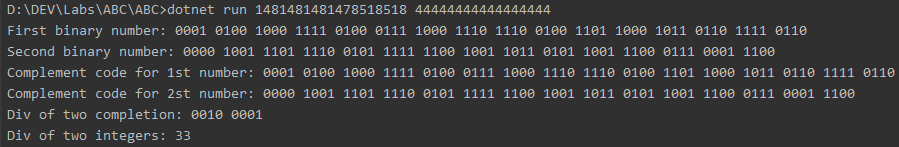


Рисунок 4. Скриншот для «a = 1481481481478518518, b = 44444444444444444»

Ответ: Результат деления = 33.

3.2.8. Тест для «a = -999999999, b = 3333»

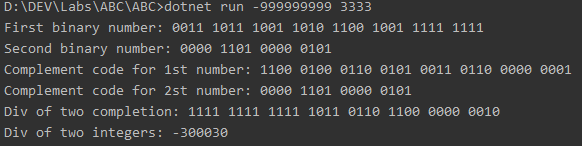


Рисунок 4. Скриншот для «a = -999999999, b = 3333»

Ответ: Результат деления = -300030.

3.2.9. Тест для «a = -999999999, b = -3333»

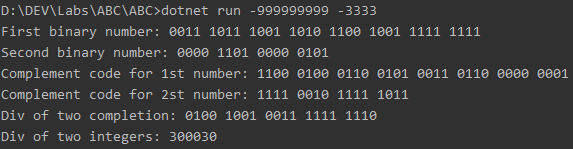


Рисунок 4. Скриншот для «a = -999999999, b = -3333»

Ответ: Результат деления = 300030.

3.2.10. Тест для «a = 999999999, b = -3333»

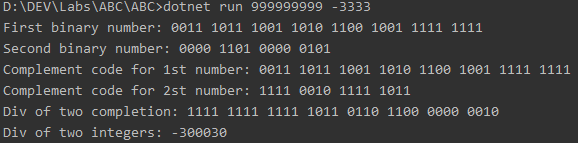


Рисунок 4. Скриншот для «a = 999999999, b = -3333»

Ответ: Результат деления = -300030.

3.2.6. Резюме.

В своих примерах я старалась рассмотреть разные случаи. Были рассмотрено деление двух положительных чисел, двух отрицательных чисел, отрицательного и положительного чисел, сделана проверка на некорректный ввод.

4. Выводы

В ходе выполнения лабораторной работы я ознакомилась с понятиями прямого кода, дополнительного кода, деления в прямом и дополнительном коде, изучила различные алгоритмы деления бинарных чисел в дополнительном коде.

Так же рассмотрела работу арифметико-логического устройства (АЛУ). Для реализации использовал язык С# и интегрированную среду разработки Rider.

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/>

4. Учебное пособие / В. М. Яшин. - М.: ИНФРА-М, 2008. - 254 с.

5. Учебное пособие / Б. Н. Ковригин. – М.: МИФИ, 2007. – 40 с.

ПРИЛОЖЕНИЕ 1

Текст программы

Модуль 1. Program.cs

using System;  
using System.Collections.Generic;  
  
namespace ABC  
{  
 public static 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($"Complement code for 2st number: {new Binary(numbers[1])}");  
 try  
 {  
 Console.WriteLine($"Div of two completion: {Calculate(numbers, Operations.**Div**)}");  
 Console.WriteLine($"Div of two decimals: {numbers[0] / numbers[1]}");  
 }  
 catch (Exception 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]),  
 Operations.**Mul** => new Binary(numbers[0]) \* new Binary(numbers[1]),  
 Operations.**Div** => new Binary(numbers[0]) / new Binary(numbers[1]),  
 \_ => throw new ArgumentException($"unknown operation: {operation}")  
 };  
 }  
 }

public class Binary  
{  
 private enum BinarySystem  
 {  
 **Base** = 2,  
 **MinLen** = 4,  
 **MaxLength** = 64  
 }  
   
 private IEnumerable<int> Value { get; }  
  
 public Binary(decimal num)   
 {  
 Value = num < 0 ? ToDirectCode(-num) : ToDirectCode(num);  
 if (num < 0)  
 {  
 Value = ToComplementCode();  
 }  
 }  
  
 private 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);  
 }  
  
 private IEnumerable<int> ToComplementCode()  
 {  
 var code = this;  
 if (Value.First() != 1)  
 {  
 code = ToInvertCode() + new Binary(1);  
 code.Value.ToArray()[0] = 1;  
 }  
   
 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, binary2);  
 var res = new List<int>().ExpandBegin(item1.Count);  
 var shift = 0;  
 item2.Reverse();  
 foreach (var i in item2)  
 {  
 if (i != 0)  
 {  
 var shifted = item1.GetRange(shift, item1.Count - shift).ExpandEnd(shift);  
 res = CalculateSum(res, shifted);  
 }  
 shift++;  
 }  
  
 var range = res.IndexOf(1);  
 return new Binary(NormalizeNumber(res.GetRange(range, res.Count - range)).ToList());  
 }  
  
 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 override string ToString()  
 {  
 return Regex.Replace(string.Join("", Value), "**.{4}**", "$0 ");  
 }  
}  
  
public static class Extensions  
{  
 public static List<int> ExpandBegin(this IEnumerable<int> list, int discharge, int value = 0)  
 {  
 var expandedList = GenerateList(discharge, value).ToList();  
 expandedList.AddRange(list);  
 return expandedList;  
 }  
   
 public static List<int> ExpandEnd(this IEnumerable<int> list, int discharge, int value = 0)  
 {  
 var expandedList = new List<int>(list);  
 expandedList.AddRange(GenerateList(discharge, value));  
 return expandedList;  
 }  
  
 private static IEnumerable<int> GenerateList(int size, int value)  
 {  
 var arr = new int[size];  
 for (var i = 0; i < arr.Length; i++)  
 {  
 arr[i] = value;  
 }  
  
 return arr;  
 }  
 }  
}