Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Кафедра информатики

Отчет по лабораторной работе №1

Эмуляция АЛУ. Операции сложения и вычитания 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

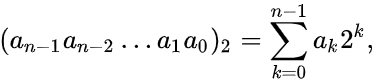
**1.Введение**

**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».

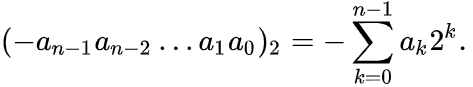
Натуральное число, записываемое в двоичной системе счисления как  {\displaystyle (a\_{n-1}a\_{n-2}\dots a\_{1}a\_{0})\_{2}}(*an – 1 an – 2 ... a1 a0*)2, имеет значение:

{\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.

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.

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»

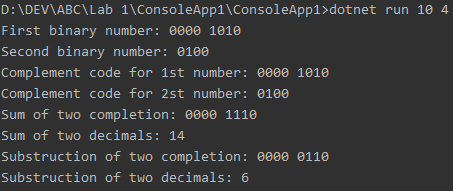


Рисунок 1. Скриншот для «a = 10, b = 4»

Ответ: Сумма = 14, Разность = 6.

3.2.2. Тест для «a = 9, b = -5»

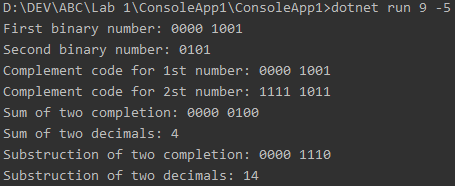


Рисунок 2. Скриншот для «a = 9, b = -5»

Ответ: Сумма = 4, Разность = 14.

3.2.3. Тест для «a = -154, b = 100»

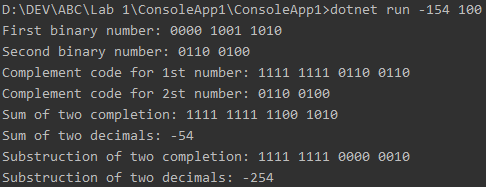


Рисунок 3. Скриншот для «a = -154, b = 100»

Ответ: Сумма = -54, Разность = -254.

3.2.4. Тест для «a = -1432, b = -104»

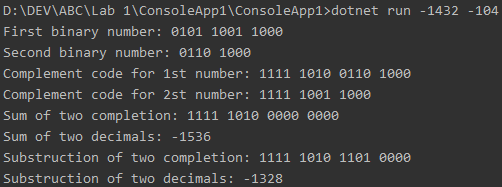


Рисунок 4. Скриншот для «a = -1432, b = -104»

Ответ: Сумма = -1536, Разность = -1328.

3.2.5. Тест для «a = 10, b = null; a = --1400, b = 23; a = 12f, b = 2; a = 10, b = -0004»

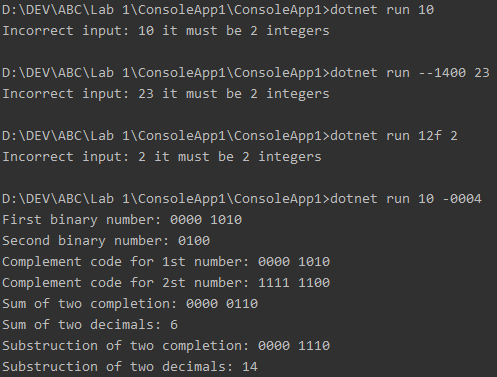


Рисунок 5. Скриншот для «a = 10, b = null; a = --1400, b = 23; a = 12f, b = 2; a = 10, b = -0004»

Ответ: Сумма = 6, Разность = 14.

3.2.6. Тест для «a =111111111111111, b = 222222222222222»

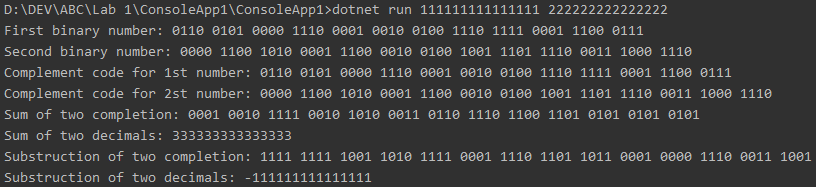


Рисунок 6. Скриншот для «a =111111111111111, b = 222222222222222»

Ответ: Сумма = 333333333333333, Разность = -111111111111111.

3.2.7. Резюме.

В своих примерах я старался рассмотреть максимально разнообразный набор случаев. Были рассмотрены сложение и разность 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 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.

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.

ПРИЛОЖЕНИЕ 1

Текст программы

Модуль 1. Program.cs

using System;   
using System.Collections.Generic;   
using System.Linq;  
using System.Text.RegularExpressions;   
  
namespace ConsoleApp1  
{  
 class Program  
 {  
 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($"Complement code for 2st number: {new Binary(numbers[1])}");  
 Console.WriteLine($"Sum of two completion: {Calculate(numbers, Operations.**Sum**)}");  
 Console.WriteLine($"Sum of two decimals: {numbers[0] + numbers[1]}");  
 Console.WriteLine($"Substruction of two completion: {Calculate(numbers, Operations.**Sub**)}");  
 Console.WriteLine($"Substruction of two decimals: {numbers[0] - numbers[1]}");  
 }  
  
 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** => Mul(numbers[0], numbers[1]),  
 \_ => throw new ArgumentException($"unknown operation: {operation}")  
 };  
 }  
  
 private static Binary Mul(decimal num1, decimal num2)  
 {  
 var res = new Binary(Math.Abs(num1)) \* new Binary(Math.Abs(num2));  
 return num1 \* num2 < 0 ? new Binary(res.ToComplementCode()) : res;  
 }  
 }  
  
 internal class Binary  
 {  
 private enum NumberSystem  
 {  
 **Base** = 2,  
 **AddedLength** = 4,  
 **MaxSystemLength** = 64  
 }  
  
 private const int **Base** = (int) NumberSystem.**Base**;  
  
 private 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 /= **Base**;  
 binary.Insert(0, Convert.ToInt32(num > Math.Truncate(num)));  
 num = Math.Truncate(num);  
 }  
  
 return NormalizeNumber(binary, 0);  
 }  
  
 private static IEnumerable<int> NormalizeNumber(IList<int> binary, int value)  
 {  
 if (binary.Count >= (int) NumberSystem.**MaxSystemLength**) return binary;  
 var binaryCount = Math.Abs(binary.Count % (int) NumberSystem.**AddedLength** - (int) NumberSystem.**AddedLength**);  
 binaryCount += binaryCount == 0 && binary[0] != 0 ? (int) NumberSystem.**AddedLength** : 0;  
  
 return binary.ExpandFromBegin(binaryCount, value);  
 }  
   
 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(NormalizeNumber(binary, 1));  
 }  
  
 public IEnumerable<int> ToComplementCode()  
 {  
 var code = ToInvertCode();  
 code += new Binary(1);  
 code.Value.ToArray()[0] = 1;  
   
 return code.Value;  
 }  
  
 public static Binary operator +(Binary binary1, Binary binary2)  
 {  
 var (item1, item2) = NormalizeLists(binary1, binary2);  
   
 return new Binary(CalculateSum(item1, item2));  
 }  
  
 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;  
 }  
  
 return list1;  
 }  
  
 public static Binary operator \*(Binary binary1, Binary binary2)  
 {  
 var (item1, item2) = NormalizeLists(binary1, binary2);  
 var res = new List<int>().ExpandFromBegin(item1.Count);  
 foreach (var i in item2)  
 {  
 res = res.ExpandFromEnd(1);  
 item1 = item1.ExpandFromBegin(1);  
 if (i != 0)  
 {  
 res = CalculateSum(res, item1);  
 }  
 }  
  
 return new Binary(NormalizeNumber(res, 0));  
 }  
  
 private static List<int> NormalizeBinary(Binary binary, int lenSub)  
 {  
 var bin = binary.Value.ToList();  
   
 return bin.ExpandFromBegin(lenSub, bin[0] == 1 && bin[1] == 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 int Add(int binItem, out int add)  
 {  
 add = 0;  
 if (binItem >= **Base**)  
 {  
 binItem -= **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> ExpandFromBegin(this IEnumerable<int> list, int discharge, int value = 0)  
 {  
 var expandedList = GenerateList(discharge, value).ToList();  
 expandedList.AddRange(list);  
 return expandedList;  
 }  
   
 public static List<int> ExpandFromEnd(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;  
 }  
 }  
}