

Министерство образования Республики Беларусь  
Учреждение образования  
«Белорусский государственный университет  
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей  
Кафедра информатики  
Дисциплина: Архитектура вычислительных систем

Отчёт по лабораторной работе №1-3  
АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ С ЦЕЛЫМИ ЧИСЛАМИ

Выполнил:  
студент гр. 153503  
Кончик Д. С.

Проверил:  
Калиновская А.А.

Минск 2023

## СОДЕРЖАНИЕ

1 Цель работы .....	3
2 Задание .....	3
3 Результат выполнения .....	8
4 Вывод.....	10

# 1 ЦЕЛЬ РАБОТЫ

Изучить принципы представления чисел в памяти компьютера, операции сложения, вычитания, умножения и деления с фиксированной точкой. Составить программу эмулятора АЛУ, реализующего соответствующие операции.

## 2 ЗАДАНИЕ

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

### Листинг 1 – Код основных функций программы

```
public class Program
{
    static void Main()
    {
        BinaryNumber a = new();
        BinaryNumber b = new();

        Console.WriteLine("Choose format of numbers: b / d");
        string? format = Console.ReadLine();

        Converter converter = format switch
        {
            "b" => MyConverter.BinaryStringToDigits,
            "d" => MyConverter.DecimalStringToDigits,
        };
        string formatString = format switch
        {
            "b" => "binary",
            "d" => "decimal"
        };
        Console.WriteLine($"\\nEnter two numbers:");
        Console.Write("a = ");
        a = new(Console.ReadLine(), converter);
        Console.Write("b = ");
        b = new(Console.ReadLine(), converter);

        Console.WriteLine($"a = {a.ToStraight()}");
        Console.WriteLine($"b = {b.ToStraight()}");
        Console.WriteLine($"a + b = {(a + b).ToStraight()}");
        Console.WriteLine($"a - b = {(a - b).ToStraight()}");
        Console.WriteLine($"a * b = {(a * b).ToStraight()}");
        (var q, var r) = a.ToExpandedBitGrid() / b;
        Console.WriteLine($"a / b = {q.ToStraight()} ({r.ToStraight()})");
    }
}
```

```

public class BinaryNumber : ICloneable
{
    private int[] _bits;
    public BinaryNumberState State { get; private set; }
    public int Length => _bits.Length;

    public BinaryNumber(int numberLength = Constants.BinaryNumberLength)
    {
        State = BinaryNumberState.Straight;
        _bits = new int[numberLength];
    }

    public BinaryNumber(string? number, Converter converter, int numLen = 32)
        : this(numberLength)
    {
        _bits = converter(number, numLen);
    }

    public BigInteger ToDecimal()
    {
        BinaryNumber tempNumber = this.ToStraight();
        BigInteger dec = 0;
        for (int i = 0; i < tempNumber._bits.Length - 1; i++)
        {
            BigInteger pow2 = i switch
            {
                0 => 1,
                _ => (BigInteger)2 << (i - 1)
            };

            dec += tempNumber._bits[i] * pow2;
        }
        if (tempNumber._bits.Last() == 1)
        {
            dec = -dec;
        }
        return dec;
    }

    private static int[] Negation(int[] digits, int length)...

    private static BinaryNumber _numberWithCertainState(BinaryNumber number,
        BinaryNumberState state)...

    public BinaryNumber ToNegative()
    {
        BinaryNumber result = ((BinaryNumber)this.Clone()).ToAdditional();
        result._bits = Negation(result._bits, result._bits.Length);
        return _numberWithCertainState(result, this.State);
    }

    public BinaryNumber ToAdditional()
    {
        var result = (BinaryNumber)this.Clone();
        if (!(State == BinaryNumberState.Additional ||
            _bits.Last() == 0))
        {
            result._bits = Negation((int[])_bits.Clone(), _bits.Length - 1);
        }
        result.State = BinaryNumberState.Additional;
        return result;
    }
}

```

```

public BinaryNumber ToStraight()
{
    var result = (BinaryNumber)this.Clone();
    if (!(State == BinaryNumberState.Straight ||
        _bits.Last() == 0))
    {
        result._bits = Negation((int[])_bits.Clone(), _bits.Length - 1);
    }
    result.State = BinaryNumberState.Straight;
    return result;
}

public BinaryNumber ToExpandedBitGrid()...
public override string? ToString()...
public object Clone()...

public static BinaryNumber operator +(BinaryNumber a, BinaryNumber b)
{
    BinaryNumber result = new() { State = BinaryNumberState.Additional };
    a = a.ToAdditional();
    b = b.ToAdditional();
    int transfer = 0;
    for (int i = 0; i < result._bits.Length; i++)
    {
        result._bits[i] = a._bits[i] + b._bits[i] + transfer;
        transfer = result._bits[i] / 2;
        result._bits[i] %= 2;
    }
    if (a._bits.Last() == b._bits.Last() &&
        result._bits.Last() != a._bits.Last())
    {
        throw new OverflowException("Overflow in \"+\" operaion.");
    }

    return result;
}

public static BinaryNumber operator -(BinaryNumber a, BinaryNumber b)
{
    try
    {
        return a.ToAdditional() + b.ToNegative().ToAdditional();
    }
    catch
    {
        throw new OverflowException("Overflow in \"-\" operaion.");
    }
}

public static BinaryNumber operator *(BinaryNumber a, BinaryNumber b)
{
    if (a.Length != b.Length)
    {
        throw new NotImplementedException();
    }

    var M = a.ToAdditional(); // Множимое
    var Q = b.ToAdditional(); // Множитель
    var A = new BinaryNumber();
    int Q_1 = 0;

    int Count = a.Length;

```

```

while (Count != 0)
{
    if (Q._bits[0] == 1 && Q_1 == 0)
    {
        A -= M;
    }
    else if (Q._bits[0] == 0 && Q_1 == 1)
    {
        A += M;
    }
    Q_1 = Q._bits[0];

    for (int i = 0; i < Q.Length - 1; i++)
    {
        Q._bits[i] = Q._bits[i + 1];
    }
    Q._bits[Q.Length - 1] = A._bits[0];

    for (int i = 0; i < A.Length - 1; i++)
    {
        A._bits[i] = A._bits[i + 1];
    }

    Count--;
}
return new BinaryNumber()
{
    State = BinaryNumberState.Additional,
    _bits = Q._bits.Concat(A._bits).ToArray()
};
}

public static (BinaryNumber, BinaryNumber) operator /(BinaryNumber a,
BinaryNumber b)
{
    if (a.Length != 2 * b.Length)
    {
        throw new InvalidDataException();
    }

    // 1.
    a = a.ToAdditional();
    b = b.ToAdditional();

    // A..Q - делимое
    var A = new BinaryNumber(b.Length).ToAdditional(); // Остаток
    A._bits = a._bits[b.Length..];
    var Q = new BinaryNumber(b.Length).ToAdditional(); // Частное
    Q._bits = a._bits[..b.Length];
    var M = b.ToAdditional(); // Делитель
    int Count = Q.Length;

    while (Count != 0)
    {
        // 2.
        for (int i = A._bits.Length - 1; i > 0; i--)
        {
            A._bits[i] = A._bits[i - 1];
        }

        A._bits[0] = Q._bits.Last();
    }
}

```

```

        for (int i = Q._bits.Length - 1; i > 0; i--)
        {
            Q._bits[i] = Q._bits[i - 1];
        }

        Q._bits[0] = 0;

        // 3.
        var A_backup = (BinaryNumber)A.Clone();

        if (M._bits.Last() == A._bits.Last())
        {
            A -= M;
        }
        else
        {
            A += M;
        }

        // 4.
        if (A_backup._bits.Last() == A._bits.Last() ||
            (A._bits.Sum() + Q._bits.Sum() == 0))
        {
            Q._bits[0] = 1;
        }
        else if (A_backup._bits.Last() != A._bits.Last() &&
            (A._bits.Sum() + Q._bits.Sum() != 0))
        {
            Q._bits[0] = 0;
            A = A_backup;
        }

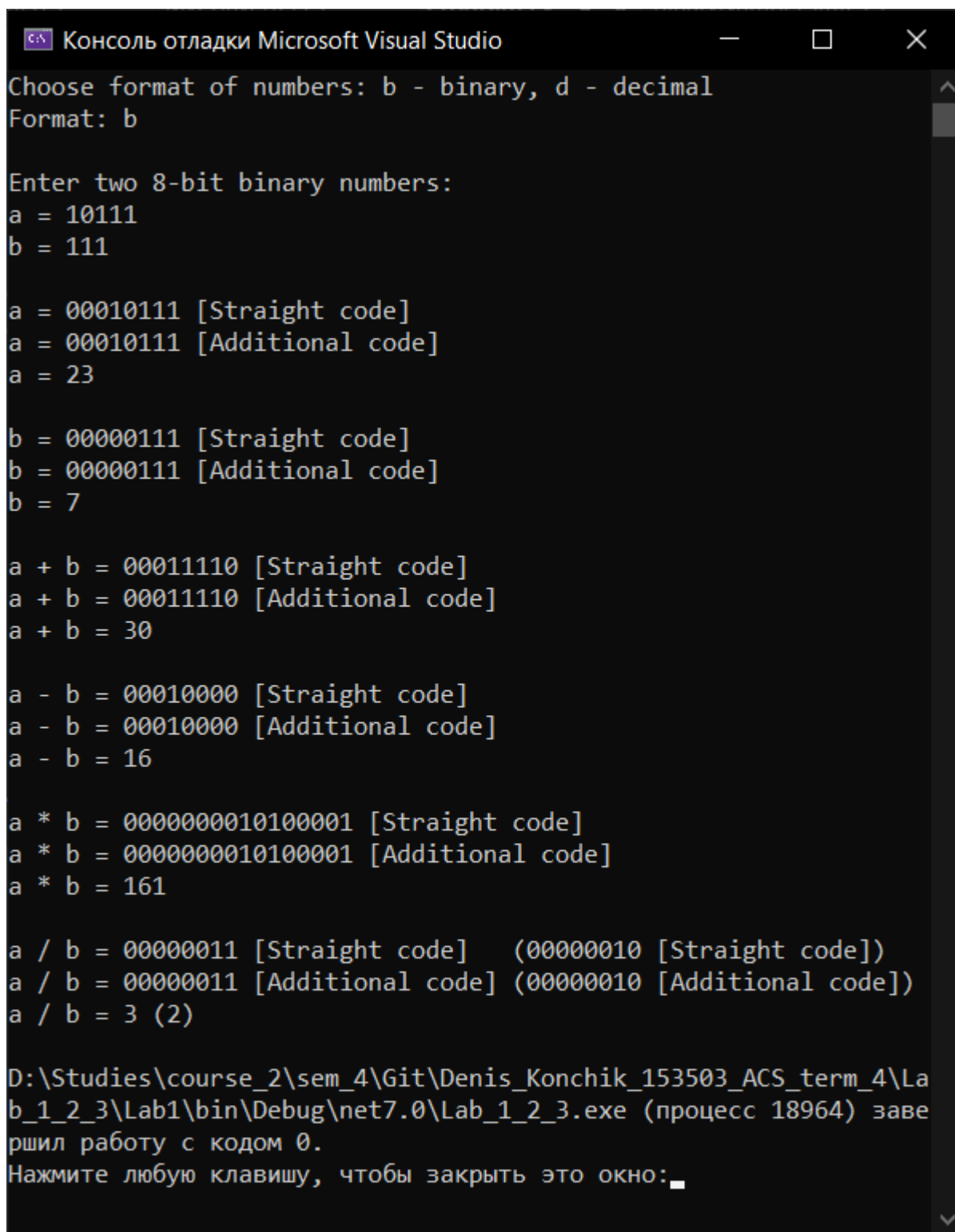
        Count--;
    }

    // 6.
    if (a._bits.Last() != b._bits.Last())
    {
        Q = Q.ToNegative();
    }

    return (Q, A);
}
}

```

### 3 РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ



```
Консоль отладки Microsoft Visual Studio
Choose format of numbers: b - binary, d - decimal
Format: b

Enter two 8-bit binary numbers:
a = 10111
b = 111

a = 00010111 [Straight code]
a = 00010111 [Additional code]
a = 23

b = 00000111 [Straight code]
b = 00000111 [Additional code]
b = 7

a + b = 00011110 [Straight code]
a + b = 00011110 [Additional code]
a + b = 30

a - b = 00010000 [Straight code]
a - b = 00010000 [Additional code]
a - b = 16

a * b = 0000000010100001 [Straight code]
a * b = 0000000010100001 [Additional code]
a * b = 161

a / b = 00000011 [Straight code] (00000010 [Straight code])
a / b = 00000011 [Additional code] (00000010 [Additional code])
a / b = 3 (2)

D:\Studies\course_2\sem_4\Git\Denis_Konchik_153503_ACS_term_4\Lab_1_2_3\Lab1\bin\Debug\net7.0\Lab_1_2_3.exe (процесс 18964) завершил работу с кодом 0.
Нажмите любую клавишу, чтобы закрыть это окно: _
```

Рисунок 1 – Ввод чисел в двоичной системе счисления



```
Консоль отладки Microsoft Visual Studio
Choose format of numbers: b - binary, d - decimal
Format: d

Enter two 8-bit decimal numbers:
a = 36
b = 75

a = 00100100 [Straight code]
a = 00100100 [Additional code]
a = 36

b = 01001011 [Straight code]
b = 01001011 [Additional code]
b = 75

a + b = 01101111 [Straight code]
a + b = 01101111 [Additional code]
a + b = 111

a - b = 10100111 [Straight code]
a - b = 11011001 [Additional code]
a - b = -39

a * b = 0000101010001100 [Straight code]
a * b = 0000101010001100 [Additional code]
a * b = 2700

a / b = 00000000 [Straight code] (00100100 [Straight code])
a / b = 00000000 [Additional code] (00100100 [Additional code])
a / b = 0 (36)

D:\Studies\course_2\sem_4\Git\Denis_Konchik_153503_ACS_term_4\Lab_1_2_3\Lab1\bin\Debug\net7.0\Lab_1_2_3.exe (процесс 20004) завершил работу с кодом 0.
```

Рисунок 2 – Ввод чисел в десятичной системе счисления

## **4 ВЫВОД**

Таким образом, была создана программа эмулятора АЛУ, выполняющего операции сложения, вычитания, умножения и деления над двумя целыми числами. На ввод программа принимает два числа, представленных в десятичной или двоичной форме.