# Лабораторная Работа No 8

**Математические Основы Защиты Информации и Информационной Безопасности**

Хосе Фернандо Леон Атупанья | НФИмд-01-24

# Содержание



# 1. Цель работы

Ознакомиться с темой целочисленная арифметика многократной точности, используя материал, представленный в лабораторной работе № 8, и используя концепции, представленные в предыдущих работах, такие как модули, максимальный общий делитель и шифрование.

# 2. Выполнение лабораторной работы - Целочисленная арифметика многократной точности

### *Алгоритм 1 (сложение неотрицательных целых чисел)*

В этой отчете следующий код реализует, u и v - это векторы цифр чисел в базе b, w инициализируется как массив нулей с n + 1 элементами для хранения результата, включая переносимую цифру. Цикл перебирает цифры от наименее значимой до наиболее значимой (j = n, n-1, ..., 1). Вычисляет сумму цифр из u и v в позиции j, включая перенос k.

```julia
function add_large_integers(u::Vector{Int}, v::Vector{Int}, b::
    n = length(u)  # Number of digits in the numbers
    w = zeros(Int, n + 1)  # Result array (extra space for carr
    k = 0  # Initial carry

    for j in n:-1:1
        # Compute the sum of digits, including the carry
        digit_sum = u[j] + v[j] + k
        w[j + 1] = digit_sum % b  # Least significant digit of
        k = digit_sum ÷ b        # Carry for the next position
    end

    # Set the carry to the first digit (w₀)
    w[1] = k
    return w
end

# Example usage
u = [3, 4, 5]  # Represents 345 in base 10
v = [6, 7, 8]  # Represents 678 in base 10
b = 10  # Base
w = add_large_integers(u, v, b)
println("Result: ", w)  # Output the result

```

Возвращается результат, который включает в себя перенос в точке w[0] и цифры суммы.

```
julia>
Result: [1, 0, 2, 3]
```

*Алгоритм 2 (вычитание неотрицательных целых чисел)*

u и v - это векторы цифр, представляющие числа по основанию b. Предполагается, что u>v. b - это основание системы счисления. Цикл перебирает цифры от наименее значимой до наиболее значимой (j = n, n-1, ..., 1). Вычисляет разницу между цифрами u и v, скорректированную заемщиком k.

```julia
function subtract_large_integers(u::Vector{Int}, v::Vector{Int}
    n = length(u)  # Number of digits in the numbers
    w = zeros(Int, n)  # Result array
    k = 0  # Initial borrow

    for j in n:-1:1
        # Compute the difference, including the borrow
        digit_diff = u[j] - v[j] + k
        if digit_diff < 0
            digit_diff += b  # Adjust with base if negative
            k = -1           # Borrow for the next position
        else
            k = 0            # No borrow needed
        end
        w[j] = digit_diff  # Store the result
    end

    return w
end

# Example usage
u = [4, 5, 6]  # Represents 456 in base 10
v = [1, 2, 3]  # Represents 123 in base 10
b = 10  # Base
w = subtract_large_integers(u, v, b)
println("Result: ", w)  # Output the result
```

Результат w возвращается в виде вектора цифр, представляющего разницу u−v.

```
Result: [3, 3, 3]
```

*Алгоритм 3 (умноджение неотрицательных целых чисел)*

u и v - это массивы, представляющие цифры чисел по основанию b, а b - это основание системы счисления. Выполняет перебор цифр v от наименее значимых до наиболее значимых (j = m, m-1, ..., 1). Пропускает вычисление, если v[j]=0, поскольку это не влияет на результат.

```julia
function multiply_large_integers(u::Vector{Int}, v::Vector{Int}
    n = length(u)  # Number of digits in u
    m = length(v)  # Number of digits in v
    w = zeros(Int, n + m)  # Result array (enough space for the

    for j in m:-1:1
        if v[j] == 0
            continue  # Skip if the digit in v is 0
        end

        k = 0  # Initial carry
        for i in n:-1:1
            t = u[i] * v[j] + w[i + j] + k  # Multiply, add to
            w[i + j] = t % b  # Store least significant digit
            k = t ÷ b         # Carry for the next iteration
        end

        w[j] = k  # Store remaining carry
    end

    return w
end
```

Результат w возвращается в виде массива цифр, представляющих продукт.

```
Result: [0, 5, 5, 3, 5]
```

*Алгоритм 4 (быстрый столбик)*

Входные: u и v: Векторы, представляющие разряды чисел с основанием b. b: Основание системы счисления. Инициализация: w: Результирующий массив, инициализированный нулями, размером n+m, достаточным для хранения продукта. t: Промежуточная сумма, инициализированная равным 0. Сохраняет весь оставшийся перенос t в w[1].

```julia
function fast_column_multiply(u::Vector{Int}, v::Vector{Int}, b
    n = length(u)  # Number of digits in u
    m = length(v)  # Number of digits in v
    w = zeros(Int, n + m)  # Result array
    t = 0  # Intermediate sum

    for s in 0:(m + n - 2)
        t = 0  # Reset intermediate sum for each position
        for i in 0:s
            # Include valid products where indices align
            if n - i > 0 && m - (s - i) > 0
                t += u[n - i] * v[m - (s - i)]
            end
        end
        # Compute result digit and carry
        w[m + n - 1 - s] = t % b
        t = t ÷ b
    end

    # Handle the final carry
    w[1] = t
    return w
end
```

Для u=123 и v=45 алгоритм вычисляет: w=[0,5,5,3,5], что соответствует 5535, произведению 123 на 45.

```
Result: [0, 3, 2, 5, 0]
```

*Алгоритм 5 (деление многоразрядных целых чисел)*

```julia
function divide_large_integers(u::Vector{Int}, v::Vector{Int},
    n = length(u) - 1  # Degree of dividend u
    t = length(v) - 1  # Degree of divisor v
    q = zeros(Int, n - t + 1)  # Quotient array
    r = copy(u)  # Copy of u, will hold the remainder

    # Step 2: Adjust for high powers of b
    while compare_large_integers(r, v, n - t, b) >= 0
        q[end] += 1
        r = subtract_scaled(r, v, n - t, b)
    end

    # Step 3: Perform the main division algorithm
    for i in n:-1:(t + 1)
        if r[i + 1] >= v[t + 1]
            q[i - t - 1] = b - 1
        else
            q[i - t - 1] = (r[i + 1] * b + r[i]) ÷ v[t + 1]
        end

        while q[i - t - 1] * (v[t + 1] * b + v[t]) >
              (r[i + 1] * b^2 + r[i] * b + r[i - 1])
            q[i - t - 1] -= 1
        end

        # Subtract q[i-t-1] * v * b^(i-t-1) from r
        r = subtract_scaled(r, v, i - t - 1, b, q[i - t - 1])
        if r[1] < 0
            r = add_scaled(r, v, i - t - 1, b)
            q[i - t - 1] -= 1
        end
    end
```

## 3. Выводы

В заключение хотелось бы отметить, что разработанные алгоритмы для многоточной целочисленной арифметики демонстрируют надежные решения для таких фундаментальных операций, как сложение, вычитание, умножение и деление. Каждый алгоритм был тщательно разработан и реализован, чтобы справиться со сложностями послойных вычислений в любой базовой системе счисления. Корректность этих методов обеспечивается строгим соблюдением принципов модульной арифметики и управлением переносами, заимствованиями и промежуточными результатами.

Алгоритмы сложения и вычитания эффективно справляются с переносами и заимствованиями из нескольких цифр, обеспечивая точность. Алгоритмы умножения, включающие как стандартный метод столбцов, так и оптимизированный метод быстрых столбцов, эффективно вычисляют результаты при минимизации вычислительных затрат. Алгоритм деления обеспечивает точные вычисления частного и остатка с помощью итеративных методов уточнения и масштабирования.

Эти реализации являются не только свидетельством нашего понимания вычислительной математики, но и основой для приложений в криптографии, численном анализе и информатике, где точность и эффективность имеют первостепенное значение.