

**МИНОБРНАУКИ РОССИИ**  
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**  
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**  
**Кафедра информационной безопасности**

**ОТЧЕТ**  
**по учебной практике**  
**Тема: Основы исследования функциональности программного**  
**обеспечения и разработки функциональных аналогов**

Студент гр. 0361

\_\_\_\_\_

Бушуев Д.И.

Руководитель

\_\_\_\_\_

Халиуллин Р.А.

Санкт-Петербург

2022

## ЗАДАНИЕ НА УЧЕБНУЮ ПРАКТИКУ

Студент Бушуев Д.И.

Группа 0361

Тема практики: наименование темы

Задание на практику: Изучить функциональность приложения, полученного в рамках задания по учебной практике, с помощью инструментальных средств реверс-инжиниринга и реализовать функциональные аналоги на ассемблере и на языке программирования высокого уровня С или С++. Вариант задания для учебной практики: 27.

Сроки прохождения практики: 29.06.2022 – 12.07.2022

Дата сдачи отчета: 12.07.2022

Дата защиты отчета: 12.07.2022

Студент

\_\_\_\_\_

Бушуев Д.И.

Руководитель

\_\_\_\_\_

Халиуллин Р.А.

## **АННОТАЦИЯ**

Задачей практической работы является создание аналога для приложения, полученного в рамках задания на ассемблере и языке C. Приложение должно совпадать с исходными выходными данными при определенных вводимых значениях.

## **SUMMARY**

The task of the practical work is to create an analogue for the application obtained in the framework of the assignment in assembly language and C language. The application must match the original output data for certain input values.

## СОДЕРЖАНИЕ

	Введение	5
1.	Анализ функциональности приложения	6
1.1.	Демонстрация работы исходного исполняемого файла	6
1.2.	Анализ функционала программы с помощью дизассемблера Ghidra	6
2.	Реализация функциональных аналогов	9
2.1.	Описание функциональности изученного приложения	9
2.2.	Описание инструкций, использованных при реализации функционального аналога на ассемблере	9
3.	Результаты тестирования реализованных функциональных аналогов	13
	Заключение	16
	Список использованных источников	17
	Приложение 1. Исходный код функционального аналога на ассемблере	18
	Приложение 2. Исходный код функционального аналога на языке программирования C	21

## **ВВЕДЕНИЕ**

Основной задачей практической работы является создание функционального аналога данного приложения.

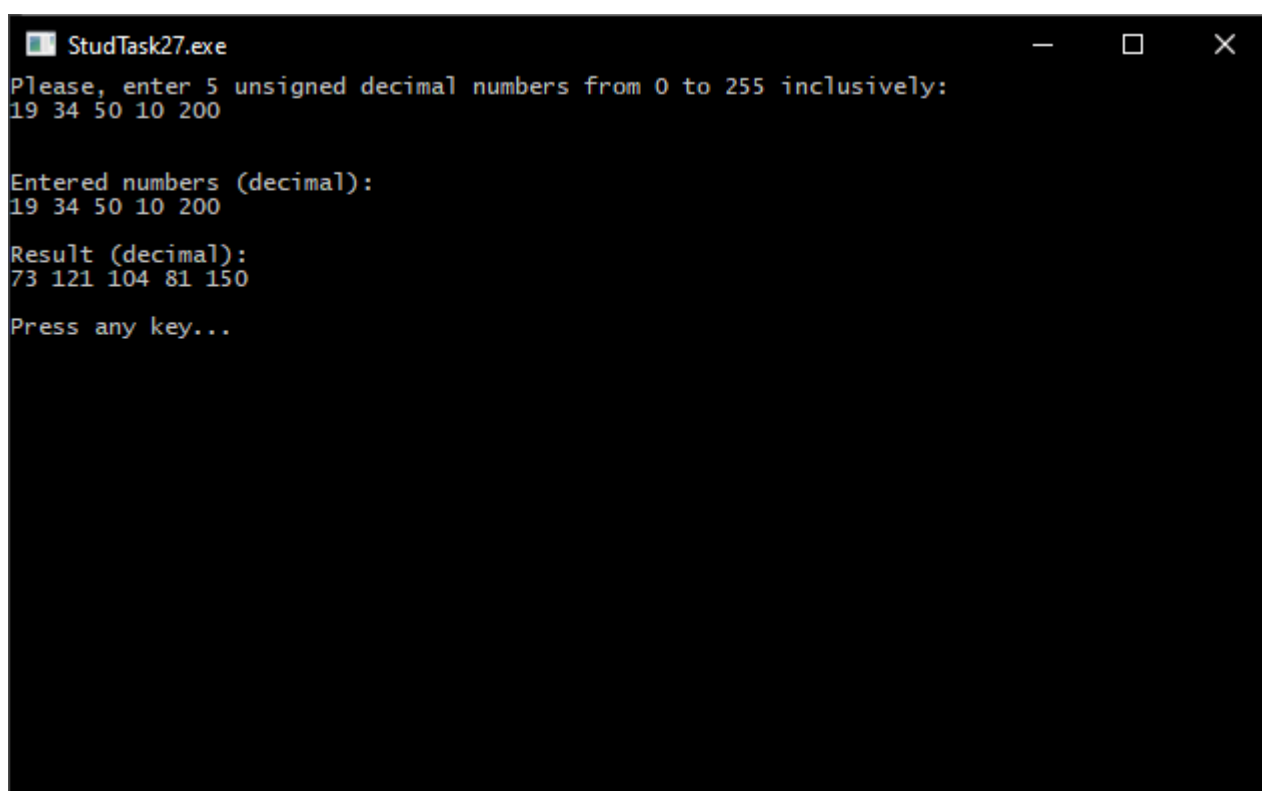
Для успешного выполнения работы мне потребовалось воспользоваться бесплатным ПО FASM, Ghidra, а также x32dbg. В результате выполнения практической работы было получены знания и опыт реверс-инжиниринга.

## 1. АНАЛИЗ ФУНКЦИОНАЛЬНОСТИ ПРИЛОЖЕНИЯ

### 1.1. Демонстрация работы исходного исполняемого файла

Заходим в папку StudTask2022 и выбираем файл по варианту, в нашем случае StudTask27.exe.

Запускаем программу и видим на экране просьбу ввести пять беззнаковых десятичных числе от 0 до 255 включительно. После того как мы введем пять чисел, программа выведет их на экран, преобразует и выведет преобразованные числа (Рисунок 1).



```
StudTask27.exe
Please, enter 5 unsigned decimal numbers from 0 to 255 inclusively:
19 34 50 10 200

Entered numbers (decimal):
19 34 50 10 200

Result (decimal):
73 121 104 81 150

Press any key...
```

Рисунок 1 – Пример работы программы StudTask27.exe

Далее чтобы провести анализ .exe файла будем использовать дополнительное ПО.

### 1.2. Анализ функционала программы с помощью дизассемблера Ghidra

Открываем наш файл StudTask27.exe в программе Ghidra, после чего нажимаем на него внутри программы чтобы запустить дизассемблер. Ждем пока наш файл проанализируется программой и смотрим результат на экране (Рисунок 2).

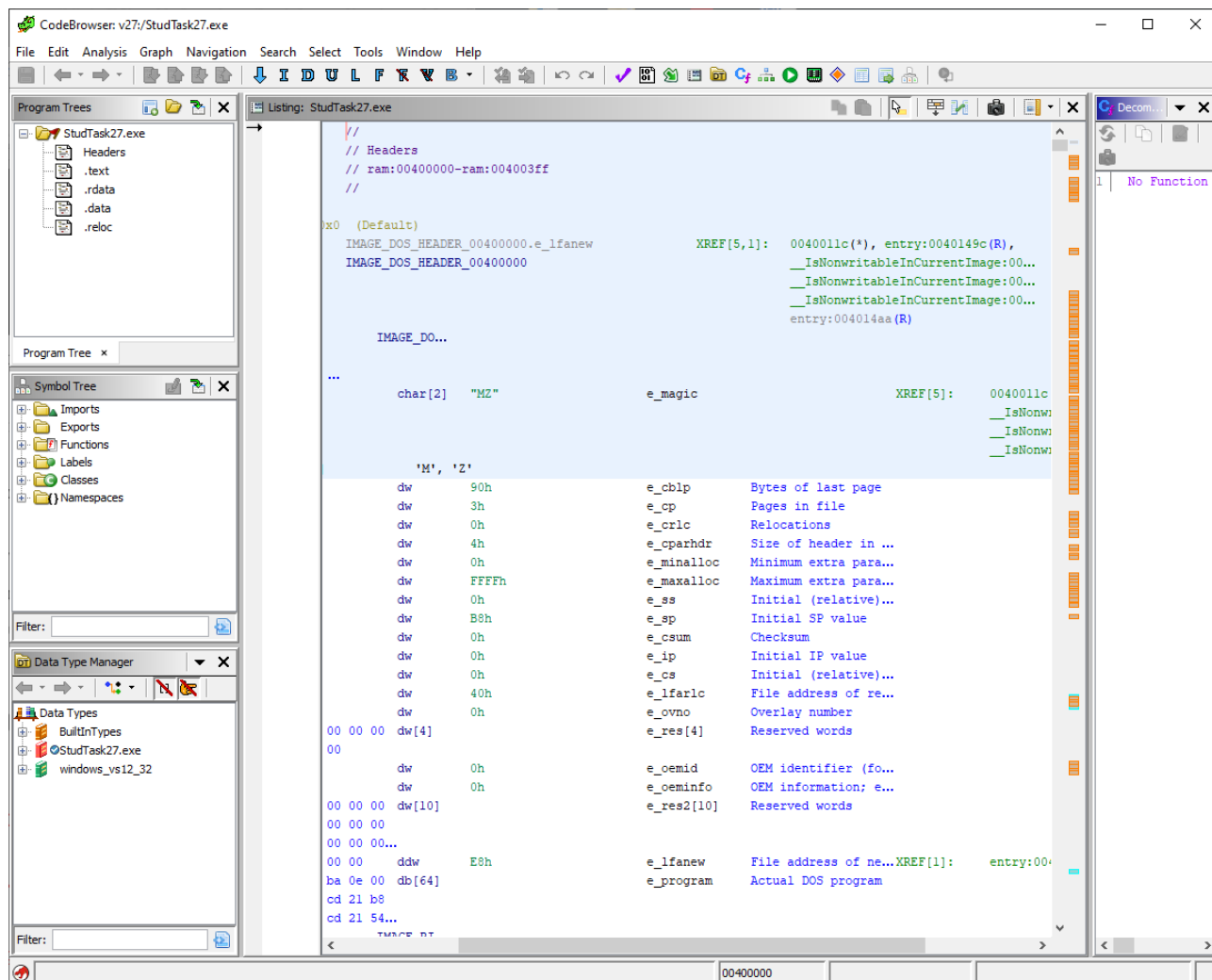


Рисунок 2 – Открытый файл в дизассемблере Ghidra

На экране мы видим "MZ" характерная сигнатура для файлов формата .exe. Чтобы найти в каком месте программы происходят преобразования над введенными числами нужно воспользоваться инструментом в программе под названием: Ghidra Defined Strings. С помощью него найти строку, которая выводится программой. После чего посмотреть в какой функции эта строка находится (FUN\_00401000) и посмотреть функции, вызываемые внутри, которые выполняют преобразования чисел.

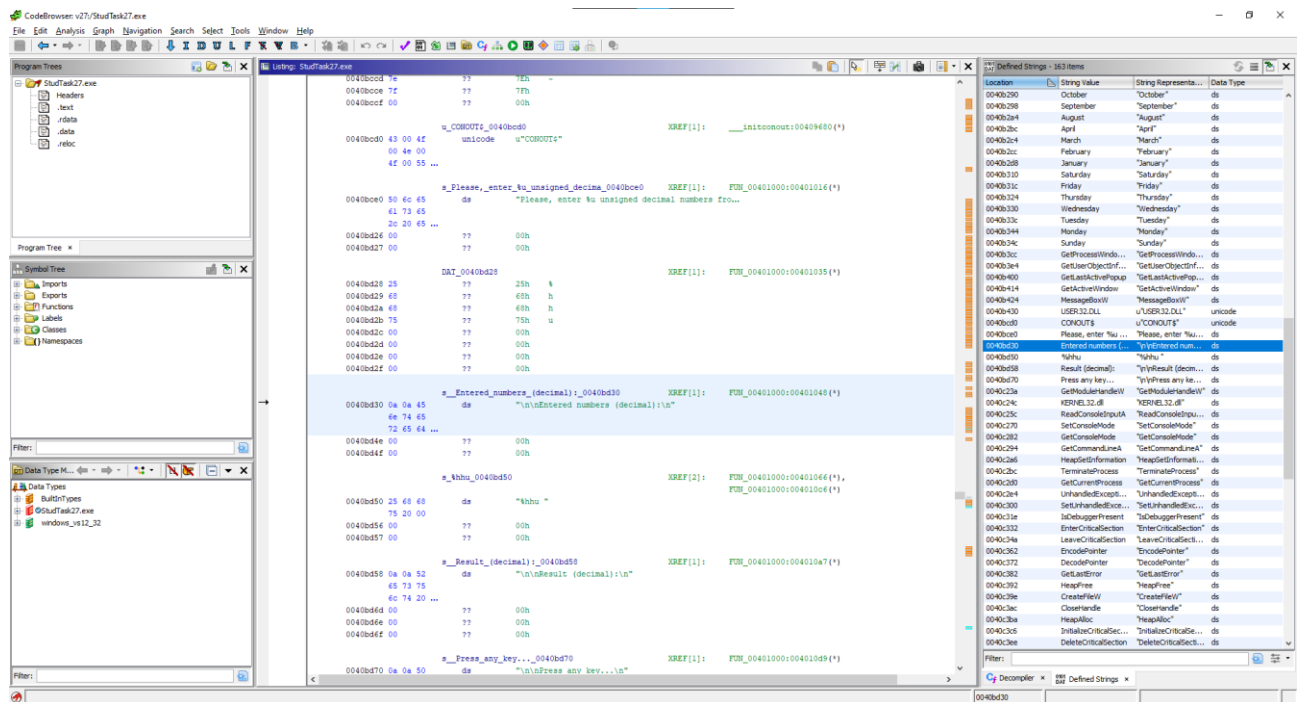


Рисунок 3 – Найденная строка

После анализа функции FUN\_00401000 было получено что весь код программы находится в ней. Код, преобразующий входные данные представлен на рисунке 4.

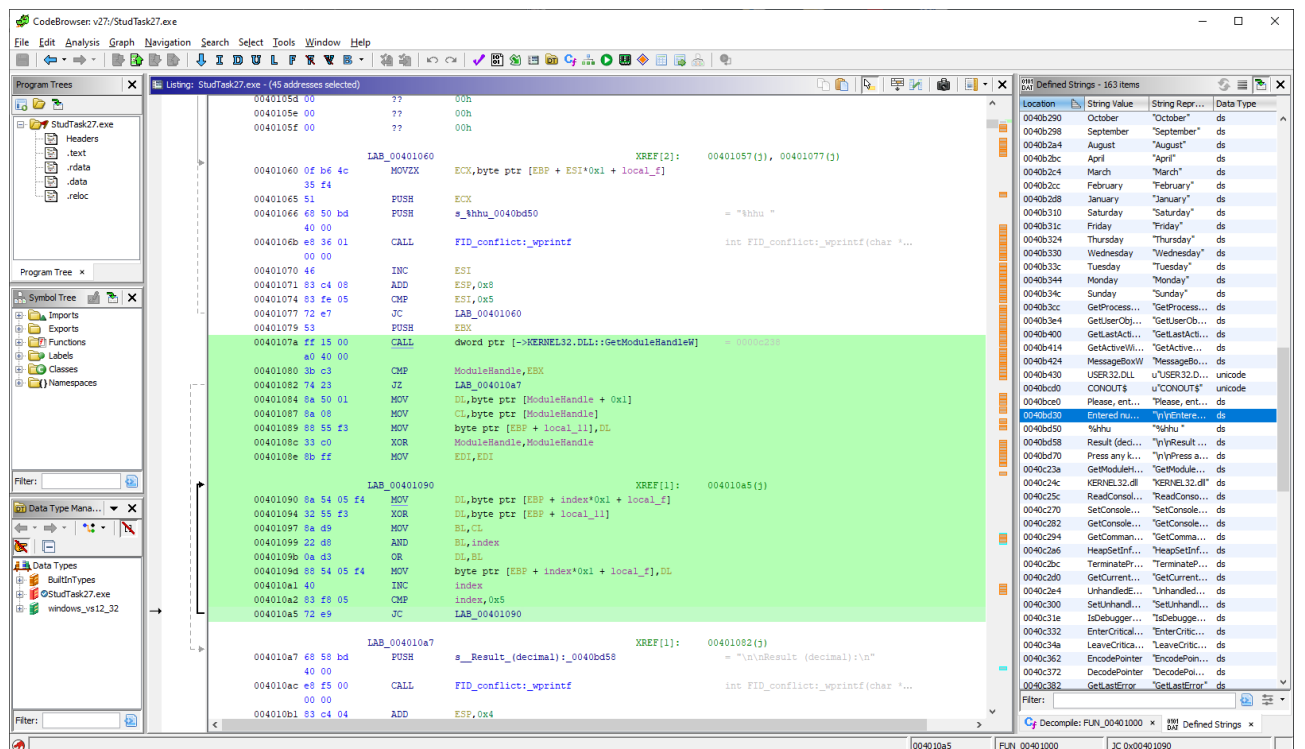


Рисунок 4 – Код, преобразующий входные данные



## 2. РЕАЛИЗАЦИЯ ФУНКЦИОНАЛЬНЫХ АНАЛОГОВ

### 2.1. Описание функциональности изученного приложения

Программа просит на вход 5 чисел размером 1 байт. В функции scanf, считывающей символы с клавиатуры, используется спецификатор %hhu, который соответствует типу данных unsigned char, размером 1 байт.

После отправки входных данных, программа выведет их на экран и далее совершит преобразования над входными данными, описанные ниже.

На рисунке 4 можно увидеть, что преобразования над числами происходят следующим образом:

$$x[i] = x[i] \oplus 0x5A + 0x4D \times i,$$

где  $x$  – элемент массива,  $i$  – индекс элемента в массиве,  $0x4D$  и  $0x5A$  – первые два байта (MZ) .exe файла (их адрес получается с использованием функции GetModuleHandleW).

### 2.2. Описание инструкций, использованных при реализации функционального аналога на ассемблере

Для написания функционального аналога на ассемблере использовались компилятор FASM, среда разработки FreshIDE, отладчик x32dbg. При написании использовались следующие инструкции процессора архитектуры x86:

1. xor <op1>, <op2>

Команда выполняет операцию исключающего ИЛИ между всеми битами двух операндов. Результат операции записывается в первый операнд.

<op1> может быть одним из следующих:

- a. Область памяти (MEM);
- b. Регистр общего назначения (REG).

<op2> может быть одним из следующих:

- a. Область памяти (MEM);
- b. Регистр общего назначения (REG);

с. Непосредственное значение – константа (IMM).

## 2. and <op1>, <op2>

Команда выполняет логическое И между всеми битами двух операндов.

Результат записывается в первый операнд.

<op1> может быть одним из следующих:

- а. Область памяти (MEM);
- б. Регистр общего назначения (REG).

<op2> может быть одним из следующих:

- а. Область памяти (MEM);
- б. Регистр общего назначения (REG);
- с. Непосредственное значение (IMM).

## 3. or <op1>, <op2>

Команда выполняется с помощью команды OR. Эта команда выполняет логическое ИЛИ между всеми битами двух операндов. Результат записывается в первый операнд.

<op1> может быть одним из следующих:

- а. Область памяти (MEM);
- б. Регистр общего назначения (REG).

<op2> может быть одним из следующих:

- а. Область памяти (MEM);
- б. Регистр общего назначения (REG);
- с. Непосредственное значение (IMM).

## 4. mov <op1>, <op2>

Команда MOV копирует содержимое <op1> и помещает это содержимое в <op2>.

<op1> может быть одним из следующих:

- а. Область памяти (MEM);

- b. Регистр общего назначения (REG);
- c. Непосредственное значение (например, число) (IMM);
- d. Сегментный регистр (SREG).

<op2> может быть одним из следующих:

- a. Область памяти (MEM);
- b. Регистр общего назначения (REG);
- c. Сегментный регистр (SREG).

#### 5. movzx <op1>, <op2>

Копирует содержимое операнда-источника в операнд-адреса, а значение расширяется нулем. Размер преобразованного значения зависит от атрибута размера операнда.

<op1> может быть одним из следующих:

- a. Область памяти (MEM);
- b. Регистр общего назначения (REG);
- c. Сегментный регистр (SREG).

<op2> может быть одним из следующих:

- a. Область памяти (MEM);
- b. Регистр общего назначения (REG);
- c. Сегментный регистр (SREG);
- d. Непосредственное значение – константа (IMM).

#### 6. inc <op1>

Команда увеличивает число на единицу.

#### 7. cmp <op1>, <op2>

Команда используется для сравнения двух операндов. Сравнение чисел выполняется по следующему алгоритму: Из <op1> вычитается <op2> (<op1> - <op2>). Если результат равен нулю, то <op1> = <op2>. Если числа равны, то

если результат равен 0, то устанавливается флаг ZF. Остальные флаги также устанавливаются или сбрасываются в зависимости от результата.

<op1> может быть одним из следующих:

- a. Область памяти (MEM);
- b. Регистр общего назначения (REG).

<op2> может быть одним из следующих:

- a. Область памяти (MEM);
- b. Регистр общего назначения (REG);
- c. Непосредственное значение – константа (IMM).

8. jne <label>

Команда переходит на метку если не равно (флаг ZF = 0).

Используемые макросы FASM:

- 1. `invoke` – макрос, использующийся для вызова функций Windows API.
- 2. `cinvoke` – макрос, использующийся для вызова функций библиотеки `msvcrt`.

### 3. РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ РЕАЛИЗОВАННЫХ ФУНКЦИОНАЛЬНЫХ АНАЛОГОВ

Протестируем функциональные аналоги, написанные на ассемблере и языке С. На рисунке 5 показан первый тест, данной программы, программы на языке си++ и программе на ассемблере соответственно.

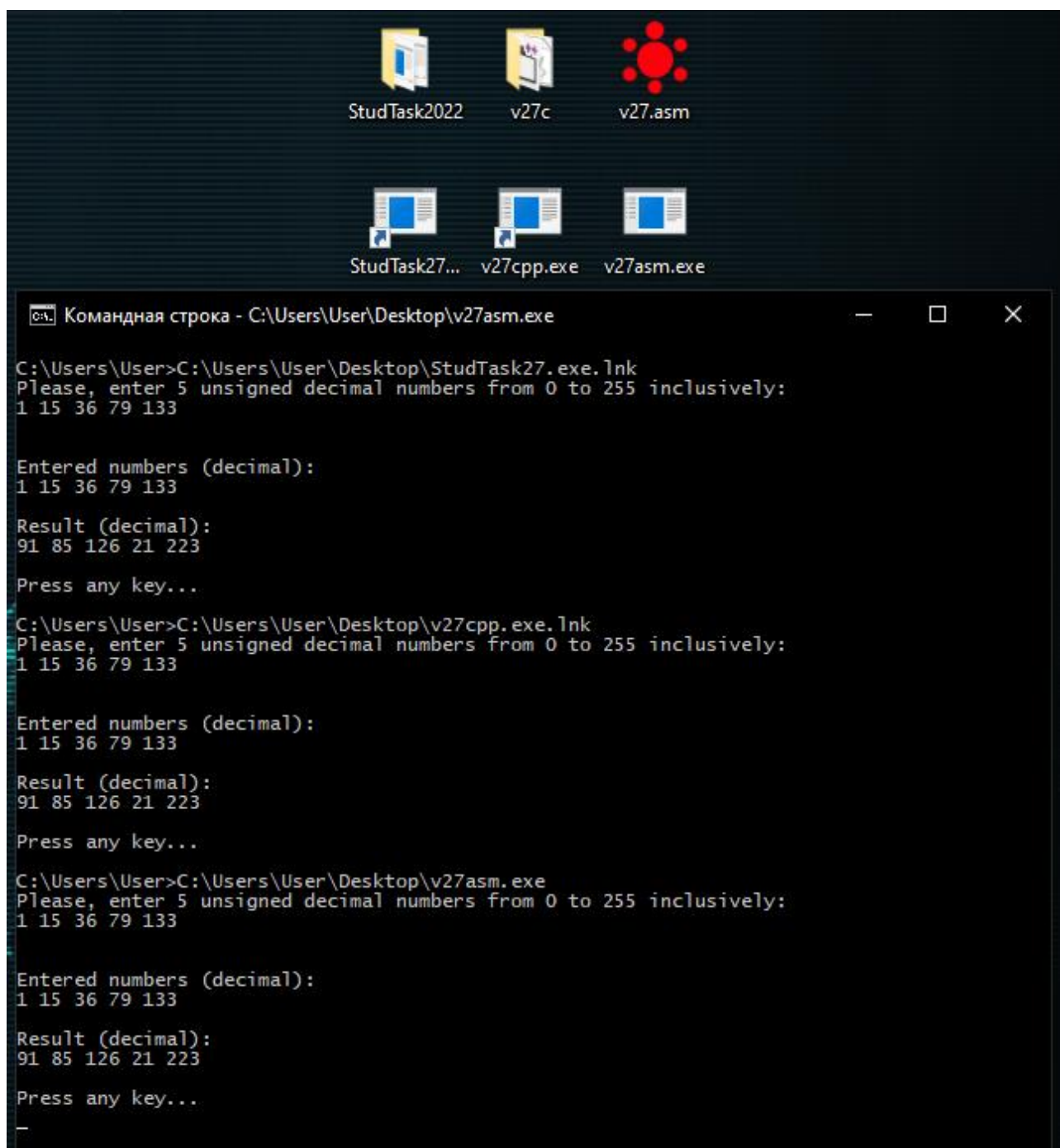


Рисунок 5 – Первый тест программы

Для большей наглядности проведем еще два теста (Рисунок 6 и Рисунок 7).

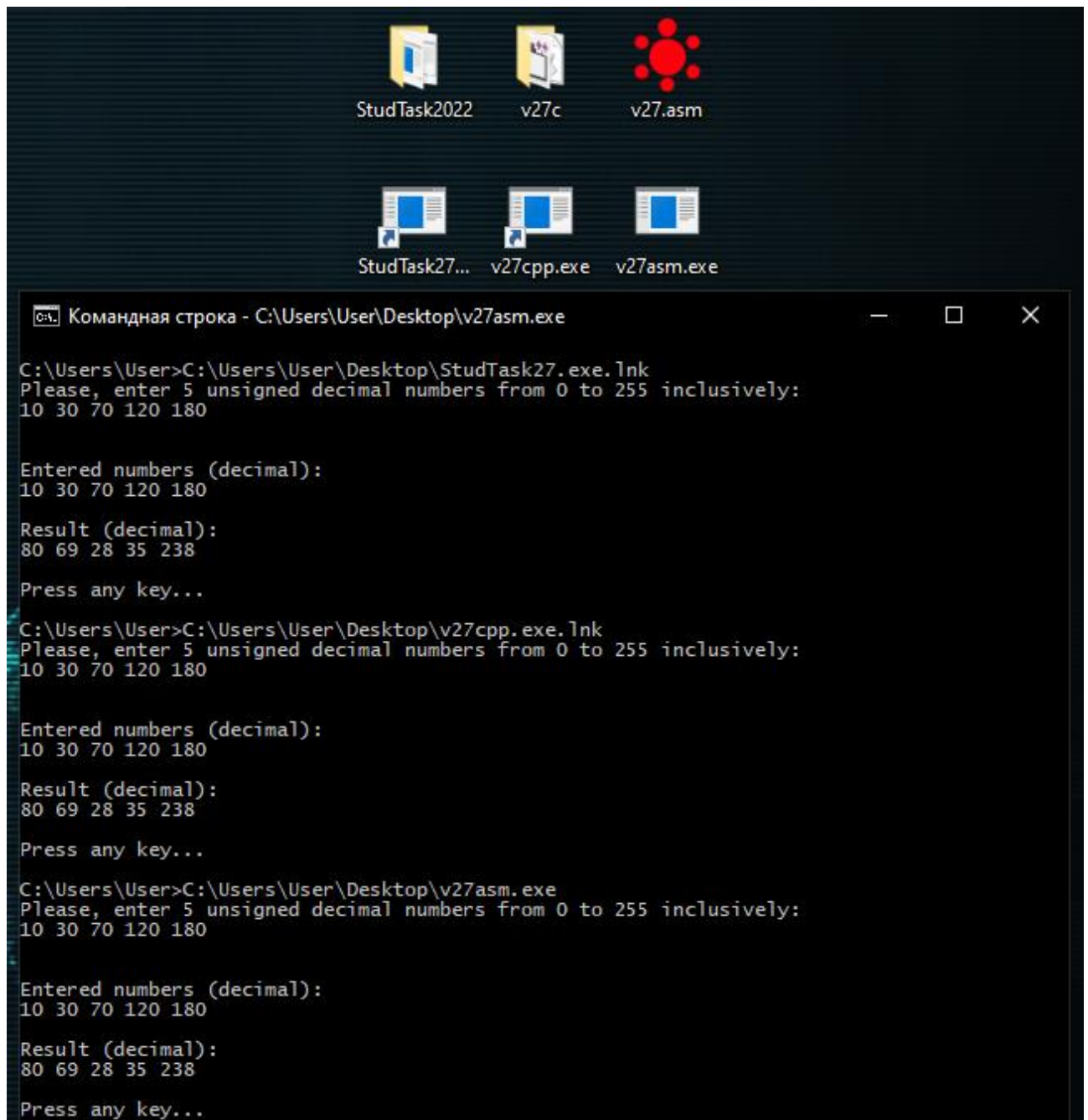


Рисунок 6 – Второй тест

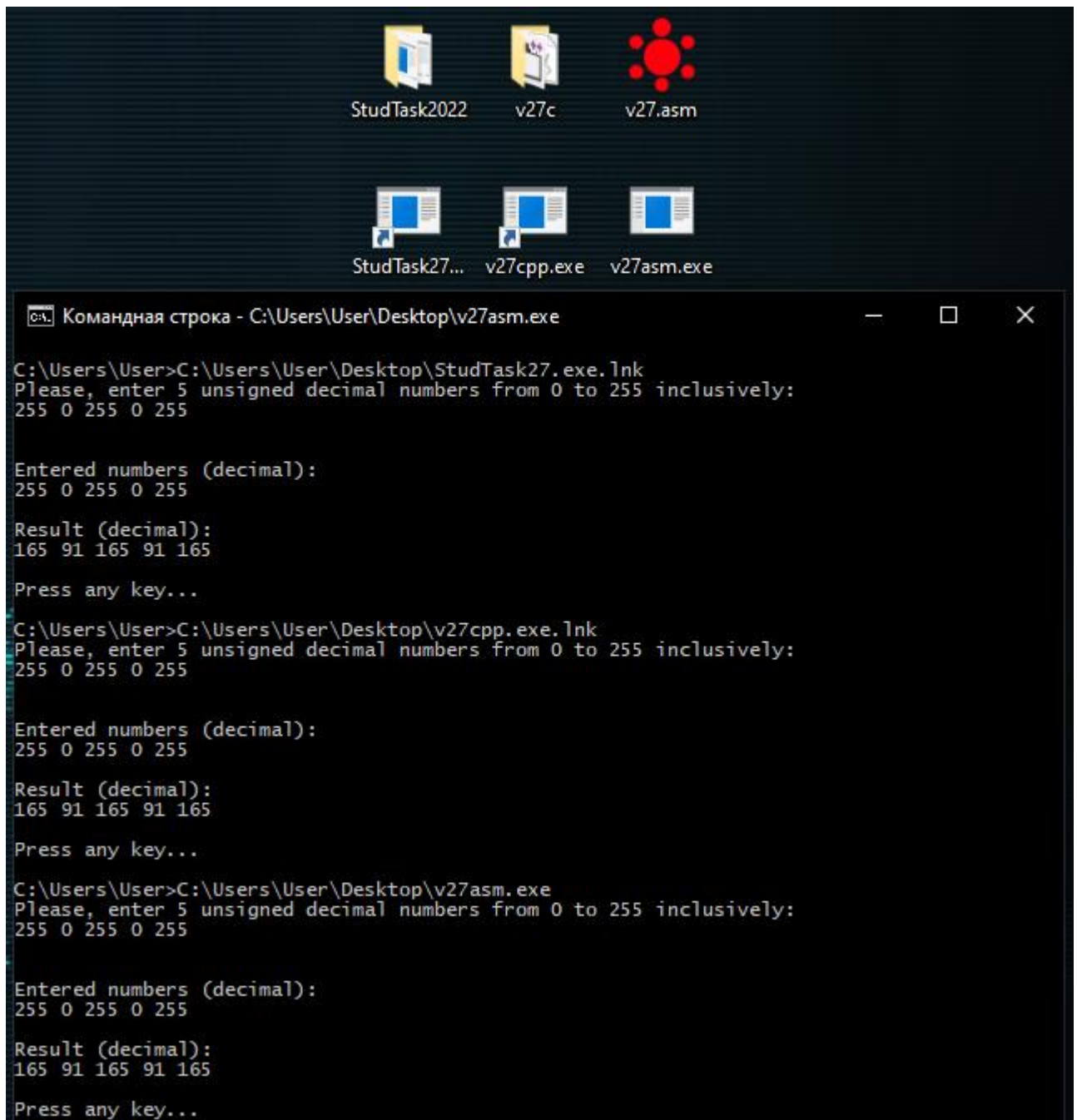


Рисунок 7 – Третий тест

По результатам трех тестов, можно сказать что функциональные аналоги соответствуют оригинальной программе и выдают те же выходные значения.

## **ЗАКЛЮЧЕНИЕ**

В результате выполнения данной практической работы на ассемблере и языке программирования С были написаны аналоги приложения, которые имеют консольный интерфейс. Программа должна при одинаковых вводных значениях выводить одинаковые выходные значения.



## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Юров В.И. Assembler: Специальный справочник. 2-е изд. СПб.: Питер, 2005. 412 с;
2. Инструкции процессоров Intel. URL: <https://av-assembler.ru/instructions/> (дата обращения: 09.07.2022).
3. Таблица с информацией о целочисленных типах// Moodle. URL: [https://vec.etu.ru/moodle/pluginfile.php/142496/mod\\_forum/post/747/About\\_C\\_Type\\_s.pdf](https://vec.etu.ru/moodle/pluginfile.php/142496/mod_forum/post/747/About_C_Type_s.pdf) (дата обращения: 09.07.2022).
4. Требование к структуре и содержанию// Moodle. URL: <https://vec.etu.ru/moodle/mod/forum/view.php?id=112650> (дата обращения: 09.07.2022).

# ПРИЛОЖЕНИЕ 1

## ИСХОДНЫЙ КОД ФУНКЦИОНАЛЬНОГО АНАЛОГА НА АССЕМБЛЕРЕ

```
FORMAT PE CONSOLE
ENTRY START
INCLUDE '%FASMINC%/WIN32WXP.INC'
SECTION '.DATA' DATA READABLE WRITEABLE
    S_GREETINGS DB 'PLEASE, ENTER 5 UNSIGNED DECIMAL NUMBERS
FROM 0 TO 255 INCLUSIVELY:', 0X0A, 0
    S_ENTER DB 'ENTERED NUMBERS (DECIMAL):', 0X0A, 0
    S_RESULT DB 'RESULT (DECIMAL):', 0X0A, 0
    S_PRESS_ANY DB 'PRESS ANY KEY...', 0X0A, 0
    S_EMPTY DB 0X0A, 0
    SP_IN DB '%HHU', 0
    SP_OUT DB '%HHU ', 0
    SP_S DB '%S', 0
    N DB 5 DUP (?)
SECTION '.TEXT' CODE READABLE EXECUTABLE
    START:
        CINVOKE PRINTF, SP_S, S_GREETINGS;PLEASE, ENTER 5
UNSIGNED DECIMAL NUMBERS FROM 0 TO 255 INCLUSIVELY:
        XOR EBX, EBX;СЧЕТЧИК
        MOV ESI, N;МАССИВ
        INPUT:;\\BВОД ЧИСЕЛ\\
        CINVOKE SCANF, SP_IN, ESI
        INC ESI
        INC EBX
        CMP EBX, 5
        JNE INPUT;////////////////
        CINVOKE PRINTF, SP_S, S_EMPTY
        CINVOKE PRINTF, SP_S, S_EMPTY
        CINVOKE PRINTF, SP_S, S_ENTER;ENTERED NUMBERS
(DECIMAL):
        XOR EBX, EBX;СЧЕТЧИК
```

```

MOV ESI, N;МАССИВ
OUTPUT_1:;\\ВЫВОД ЧИСЕЛ\\
MOVZX EAX, BYTE [ESI]
CINVOKE PRINTF, SP_OUT, EAX
INC ESI
INC EBX
CMP EBX, 5
JNE OUTPUT_1;////////////////
CINVOKE PRINTF, SP_S, S_EMPTY
;NUM[I] = NUM[I] ^ Z | M & I;TREATMENT
XOR EBX, EBX;СЧЕТЧИК
MOV ESI, N;МАССИВ
TREATMENT:;\\ОБРАБОТКА\\
MOVZX EAX, BYTE [ESI]
XOR AL, 'Z'
MOV CL, 'M'
AND CL, BL
OR AL, CL
MOV BYTE [ESI], AL
INC ESI
INC EBX
CMP EBX, 5
JNE TREATMENT;////////////////
CINVOKE PRINTF, SP_S, S_EMPTY
CINVOKE PRINTF, SP_S, S_RESULT;RESULT (DECIMAL):
XOR EBX, EBX;СЧЕТЧИК
MOV ESI, N;МАССИВ
OUTPUT_2:;\\РЕЗУЛЬТАТ\\
MOVZX EAX, BYTE [ESI]
CINVOKE PRINTF, SP_OUT, EAX
INC ESI
INC EBX
CMP EBX, 5
JNE OUTPUT_2;////////////////
CINVOKE PRINTF, SP_S, S_EMPTY

```

```

        CINVOKE PRINTF, SP_S, S_EMPTY
        CINVOKE PRINTF, SP_S, S_PRESS_ANY;PRESS ANY KEY...
        CINVOKE GETCH
        INVOKE EXITPROCESS, DWORD 0
SECTION '.IDATA' IMPORT DATA READABLE WRITEABLE
        LIBRARY KERNEL32, 'KERNEL32.DLL', \
            MSVCRT, 'MSVCRT.DLL'
        IMPORT KERNEL32, EXITPROCESS, 'EXITPROCESS'
        IMPORT MSVCRT, PRINTF, 'PRINTF', \
            GETCH, '_GETCH', \
            SCANF, 'SCANF'

```

## ПРИЛОЖЕНИЕ 2

### ИСХОДНЫЙ КОД ФУНКЦИОНАЛЬНОГО АНАЛОГА НА ЯЗЫКЕ ПРОГРАММИРОВАНИЯ C/C++

```
#DEFINE _CRT_SECURE_NO_WARNINGS
#include <IOSTREAM>
#include <STDINT.H>
#include <WINDOWS.H>
#include <CONIO.H>

INT MAIN() {
    PRINTF("%S", "PLEASE, ENTER 5 UNSIGNED DECIMAL NUMBERS FROM 0
TO 255 INCLUSIVELY:\N");
    UNSIGNED CHAR NUM[5] = { 0, 0, 0, 0, 0 };
    FOR (UNSIGNED CHAR I = 0; I < 5; I++) {
        SCANF("%HHU", &NUM[I]);
    }
    PRINTF("%S", "\N\NENTERED NUMBERS (DECIMAL):\N");
    FOR (UNSIGNED INT I = 0; I < 5; I++) {
        PRINTF("%HHU ", NUM[I]);
    }
    UINT8_T M = 0X4D, Z = 0X5A;
    FOR (UNSIGNED INT I = 0; I < 5; I++) {
        NUM[I] = (NUM[I] ^ Z) | (M & I);
    }
    PRINTF("%S", "\N\NRESULT (DECIMAL):\N");
    FOR (UNSIGNED INT I = 0; I < 5; I++) {
        PRINTF("%HHU ", NUM[I]);
    }
    PRINTF("%S", "\N\NPRESS ANY KEY...\N");
    _GETCH();
}
```