МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ

Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования
«**Московский технический университет связи и информатики**»
(**МТУСИ**)

Кафедра «Системное программирование»

# ОТЧЕТ

по лабораторной работе № 2

по дисциплине «**Системное программирование**»

Выполнил:
студент гр. БФИ2202
_____ Сидорук Д. В.
«____» _____ 2024 г.

Проверил:
старший преподаватель
_____ Шананин В. А.
«____» _____ 2024 г.

Москва, 2024 г.

**Содержание**

## 1 Цель работы

Получить базовые навыки программирования на ассемблере под архитектуру x86 с использованием NASM.

## 2 Задание

Разработать приложение для нахождения всех натуральных чисел, которые в десятичной системе счисления равны сумме своих цифр, возведенных в степень, равную количеству его цифр.

## 3 Ход работы

В листинге ниже приведен код разработанной программы (1):

Лист. 1 – Код программы

```
section .data
    format: db '%d',10

global main
section .text

extern printf

pow:
    ; Входные данные:
    ; rcx - число, которое нужно возвести в степень
    ; rdx - степень, в которую нужно возвести число
    ; Выход:
    ; rax - результат возведения rcx в степень rdx
    ; Используемые переменные:
    ; r9 - оригинальное число, которое нужно возвести в степень (начальное значение: r10)
    ; r10 - переменная, используемая для вычислений (начальное значение: rcx)
    ; r11 - степень, в которую нужно возвести число (начальное значение: rdx)
    ; r8 - счетчик, в какую степень уже возведено r10 (начальное значение: 0)

    mov r10, rcx ; int number (arg)
    mov r11, rdx ; int power (arg)
    dec r11 ; у нас уже есть число, возведенное в первую степень, так что уменьшаем степень на 1
    mov r9, r10 ; int original_number = number
    mov r8, 0 ; int i = 0
    mov rax, r10 ; промежуточный результат: оригинальное число
__condition0:
    cmp r8, r11 ; сравниванием, в какую степень возведено, и в какую степень нужно возвести
    jne __loop0 ; если они еще не равны, то идем возводить
```

```
30      ret ; иначе выходим из функции: результат уже лежит в rax
31  __loop0:
32      mov rax, r10 ; rax - то, что нужно умножить
33      mov rdx, r9 ; rdx - то, на что умножаем
34      mul rdx ; умножаем rax на rdx, результат сохраняется в rax
35      mov r10, rax ; сохраняем результат умножения в нашу переменную
36      inc r8 ; и инкрементируем счетчик, что мы посчитали степень
37      jmp __condition0 ; переходим к проверке, завершено ли
        ↪ возведение
38
39  number_of_digits:
40      ; Входные данные:
41      ; rcx - число, количество цифр в котором необходимо посчитать
42      ; Выходные данные:
43      ; rax - количество цифр в переданном числе
44      ; Используемые переменные:
45      ; r8 - копия rcx
46      ; r9 - счетчик цифр в числе r8
47      mov r8, rcx ; int number (arg)
48      mov r9, 0 ; int i = 0
49  __condition1:
50      cmp r8, 0 ; Сравниваем число с нулем: для проверки, перебрали
        ↪ ли мы все разряды
51      jne __loop1 ; Если не перебрали, идем перебирать
52      mov rax, r9 ; Иначе перемещаем r9 в rax
53      ret ; И выходим из функции
54  __loop1:
55      mov rax, r8 ; Перемещаем число в rax для того, чтобы потом его
        ↪ поделить
56      cqo ; производим расширение для дальнейшего деления
57      mov rcx, 10 ; rcx - то, на что мы делим
58      idiv rcx ; производим целочисленное деление rax на rcx
59      mov r8, rax ; обновляем значение в r8
60      inc r9 ; и обновляем счетчик, что мы посчитали очередной
        ↪ десятичный разряд
61      jmp __condition1 ; переходим к проверке, не завершен ли
        ↪ подсчет цифр
62
63  main:
64      ; Используемые переменные:
65      ; r15 - текущее число, для которого происходит вычисление
        ↪ (начальное значение: 1)
66      ; r12 - копия r15 для использования в вычислении (начальное
        ↪ значение: r15)
67      ; r13 - результат вычисления (начальное значения: 0)
68      ; r14 - количество цифр в числе r12
69
70      mov r15, 1
71
72   __loop3:
73      mov rbx, r15 ; int original_number
74      mov r12, r15 ; int number
75      mov r13, 0 ; int result
```

```
76      mov rcx, r12 ; number_of_digits ожидает свой аргумент в rcx
77      call number_of_digits ; вызываем функцию number_of_digits
78      mov r14, rax ; int i = number_of_digits(number)
79
80  __condition2:
81      cmp r12, 0 ; r12 равное 0 свидетельствует о том, что мы
        ↪ посчитали result
82      jne __loop2 ; если мы не посчитали его, нам нужно его
        ↪ досчитать
83
84      cmp rbx, r13 ; сравниваем, является ли посчитанный result
        ↪ искомым числом
85      je __print ; если является, то печатаем его
86      jmp __continue_iter ; иначе идем проверять следующее число
87
88  __print:
89      mov rdi, format ; строка, которая передается printf
90      mov rsi, r13 ; аргумент, передаваемый printf
91      mov eax, 0 ; переменное количество аргументов
92      call printf ; вызываем printf
93
94  __continue_iter:
95      inc r15 ; инкрементируем число
96      cmp r15, 0 ; если оно дошло до нуля - значит, оно
        ↪ переполнилось, и мы дальше считать не можем
97      jne __loop3 ; если не дошло - переходим к вычислению
98
99      ret ; если дошло - выходим
100 __loop2:
101     mov rax, r12 ; rax - что делим
102     cqo ; расширяем его знак для деления
103     mov rcx, 10 ; rcx - то, на что делим
104     idiv rcx ; rax - результат, rdx - остаток
105     mov r12, rax ; сохраняем результат в переменную
106
107     mov rcx, rdx ; rcx - аргумент pow - то, что возводим в степень
108     mov rdx, r14 ; rdx - аргумент pow - то, во что возводим в
        ↪ степень
109     call pow ; вызываем pow
110     add r13, rax ; добавляем результат pow к переменной result
111
112     jmp __condition2 ; переходим к сравнению: посчитан ли result?
```

На рисунке ниже представлен результат работы программы. (1)

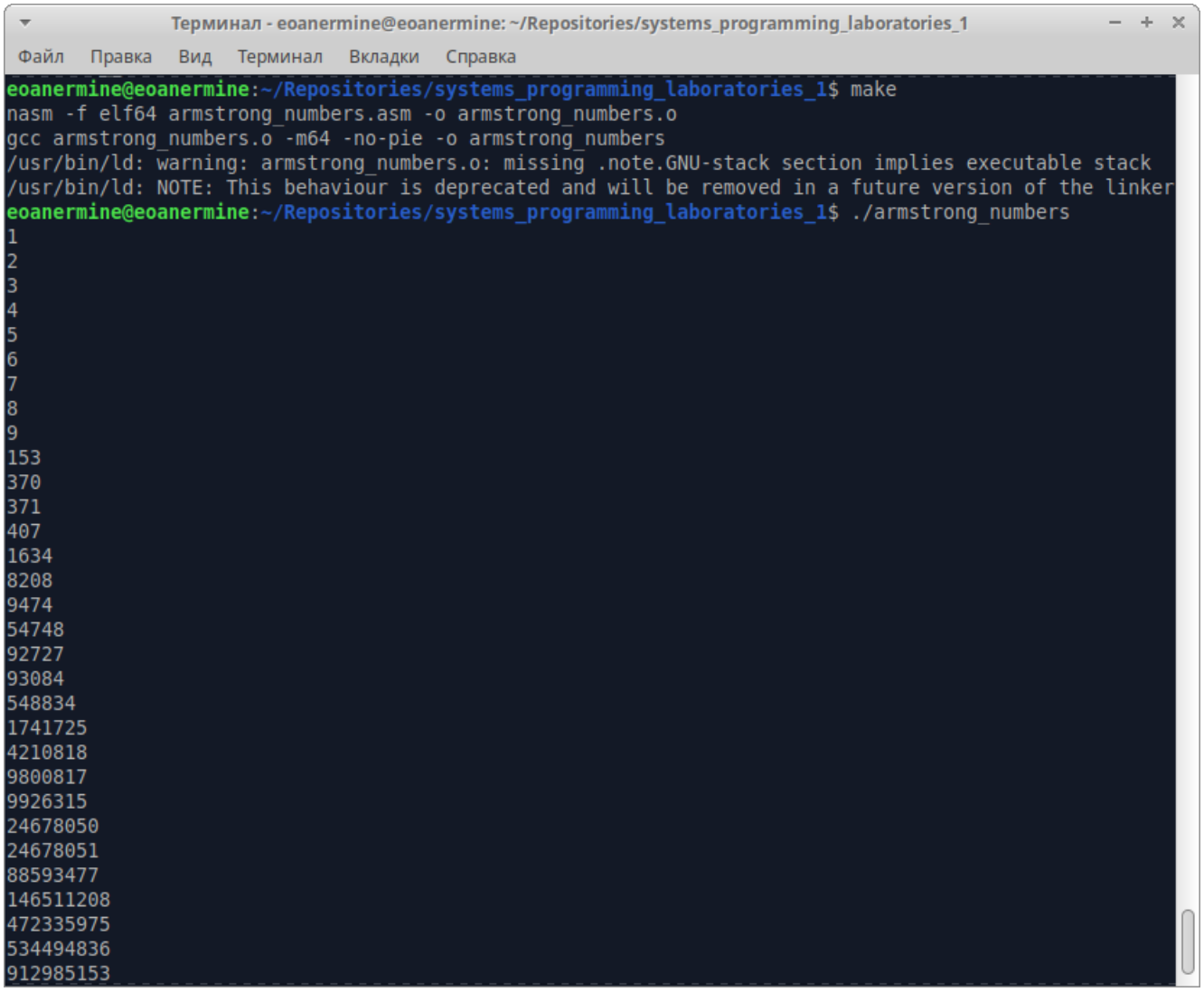


Рис. 1 – Результат работы программы

## Заключение

В ходе выполнения данной лабораторной работы мы получили базовые навыки программирования на ассемблере под архитектуру x86 с использованием NASM.