Отчёт по лабораторной работе No8

Программирование цикла. Обработка аргументов командной строки.

Карпухин Клиим

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

Цель данной лабораторной работы — приобретение навыков написания программ на языке ассемблера NASM с использованием циклов и обработкой аргументов командной строки, а также закрепление знаний об организации стека и использовании инструкций `push` и `pop` для временного сохранения значений регистров.

2 Выполнение лабораторной работы

2.1 Реализация циклов в NASM

Создал каталог для программ лабораторной работы №8, перешёл в него и создал файл `lab8-1.asm` (рис. 2.1).

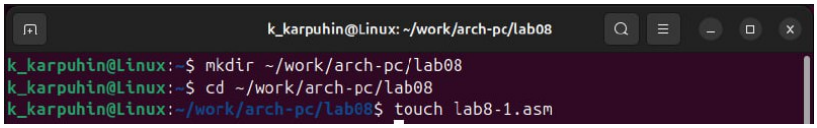

Рисунок 2.1: Создание каталога для программ лабораторной работы №8, переход в него и создание файла lab8-1.asm

Ввёл в файл `lab8-1.asm` текст программы из листинга 8.1, реализующей вывод значений регистра `ecx` с использованием инструкции `loop` (рис. 2.2).

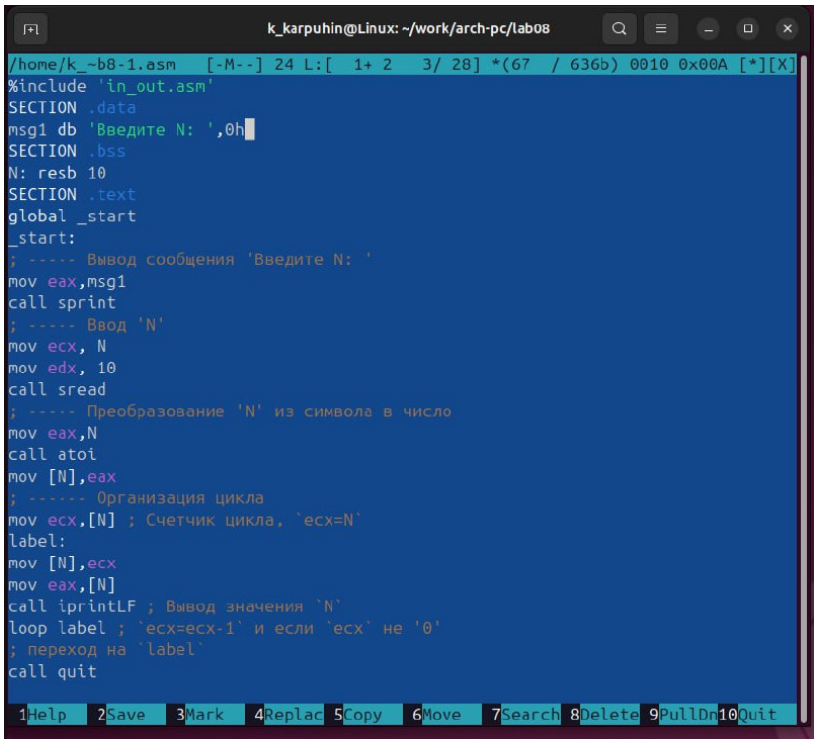

Рисунок 2.2: Введённый текст программы вывода значений регистра ecx (листинг 8.1)

Создал объектный файл и исполняемый файл, затем запустил программу (рис. 2.3).

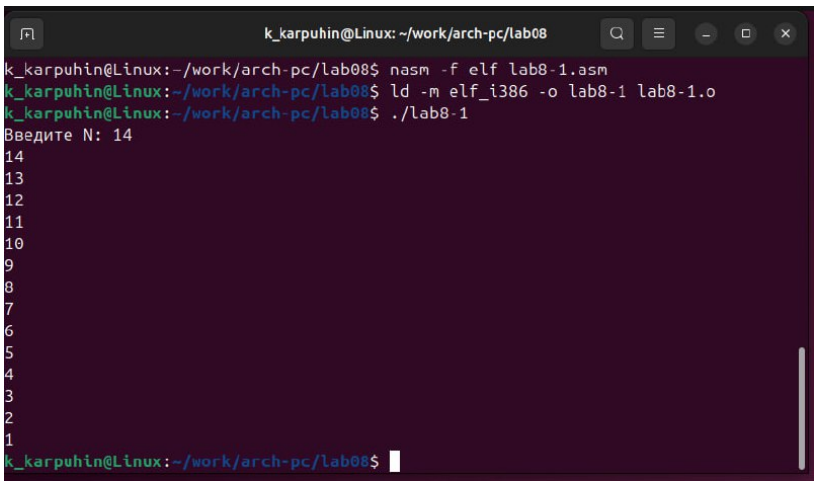

Рисунок 2.3: Создание объектного и исполняемого файла, запуск программы lab8-1

При запуске программы ввёл значение N = 14. Программа выводила

последовательность значений регистра ecx от 14 до 1, каждое значение — с новой строки.

Изменил текст программы, добавив явное изменение регистра ecx внутри тела цикла:

```
label:
    sub ecx, 1
    mov [N], ecx
    mov eax, [N]
    call iprintLF
    loop label
```

(рис. 2.4).

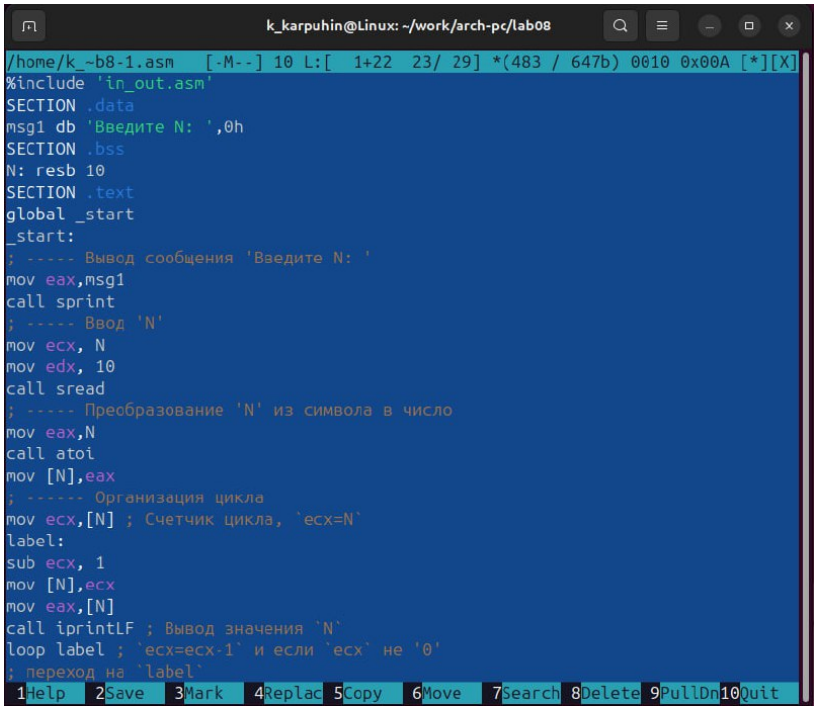

Рисунок 2.4: Изменённый текст программы с дополнительным изменением регистра ecx внутри цикла

Снова создал объектный и исполняемый файлы и запустил программу (рис. 2.5).

Рисунок 2.5: Создание и запуск исполняемого файла после изменения программы

После внесённых изменений:

- регистр `ecx` уменьшался **дважды** на каждой итерации — один раз явной командой `sub ecx, 1` и один раз внутри инструкции `loop`;
- количество проходов цикла стало меньше введённого N;
- выводились не все значения от N до 1, а только часть (каждое второе значение), и цикл завершался раньше.

Сделал вывод, что при использовании инструкции `loop` нельзя произвольно изменять регистр `ecx` внутри тела цикла, иначе нарушается корректность работы счётчика.

Для сохранения корректности работы цикла внёс изменения в текст программы, добавив использование стека: перед локальным изменением `ecx` значение счётчика сохраняется в стек командой `push`, а после выполнения тела цикла восстанавливается командой `pop`. Модифицированный фрагмент программы имеет вид:

```
label:
    push ecx
    sub ecx, 1
    mov [N], ecx
    mov eax, [N]
```

```
    call iprintLF
    pop ecx
    loop label
```

(рис. 2.6).

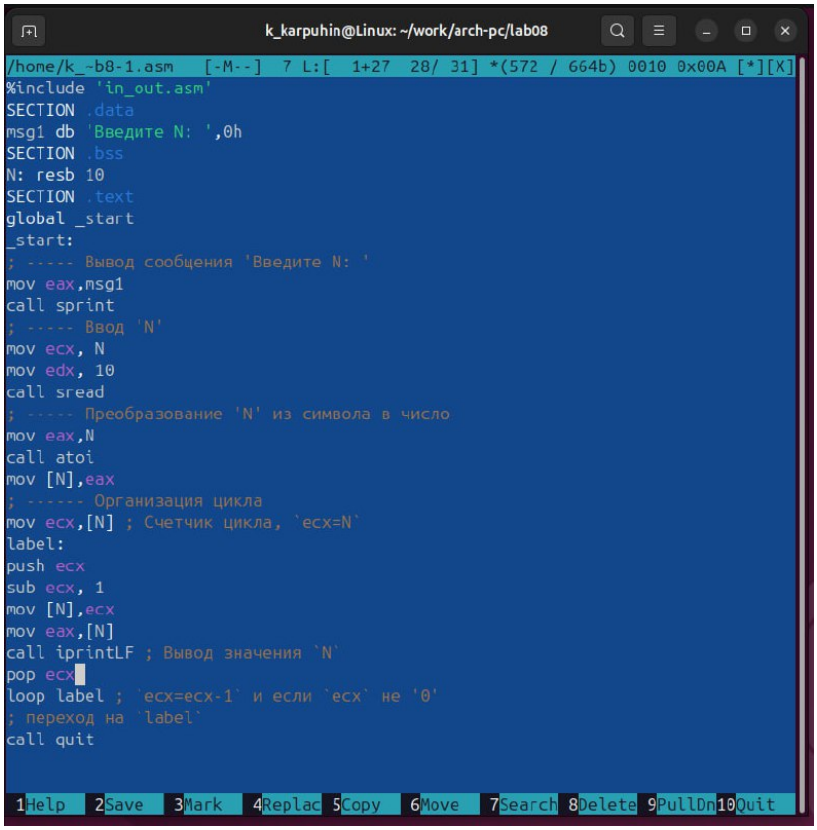

Рисунок 2.6: Изменённый текст программы с использованием стека (push/pop) для сохранения счётчика цикла

Собрал и запустил программу ещё раз (рис. 2.7).

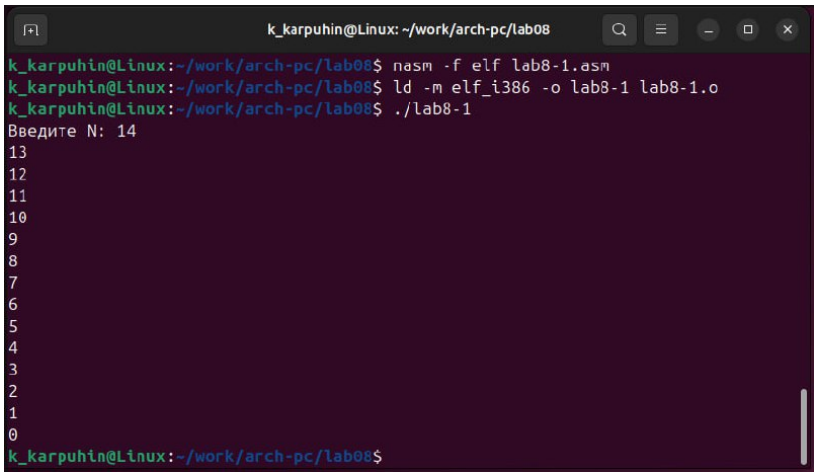

Рисунок 2.7: Создание и запуск исполняемого файла после добавления инструкций push/pop

После исправления:

- число проходов цикла снова стало равным введённому значению N;
- значения регистра `ecx` последовательно уменьшались от N до 1;
- инструкция `loop` корректно использовала восстановленное значение `ecx` как счётчик, а временные изменения внутри тела цикла не нарушали логику работы цикла.

Таким образом, было продемонстрировано, как с помощью стека можно временно изменять регистр `ecx` внутри цикла, не нарушая работу инструкции `loop`.

2.2 Обработка аргументов командной строки

2.2.1 Программа вывода аргументов командной строки

Создал файл `lab8-2.asm` в каталоге `~/work/arch-pc/lab08` и ввёл в него текст программы из листинга 8.2, реализующей вывод аргументов командной строки (рис. 2.8).

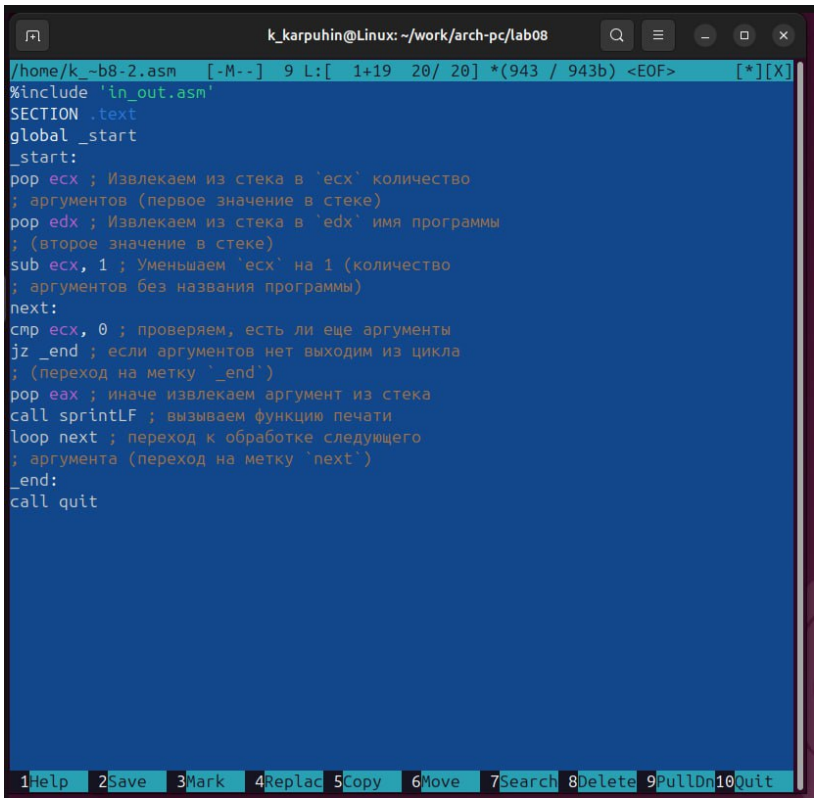

Рисунок 2.8: Текст программы lab8-2.asm, обрабатывающей аргументы командной строки (листинг 8.2)

Текст программы использует стек для получения количества аргументов и имени программы, а затем в цикле извлекает и выводит каждый аргумент командной строки.

Собрал и запустил программу, передав ей несколько аргументов (рис. 2.9).

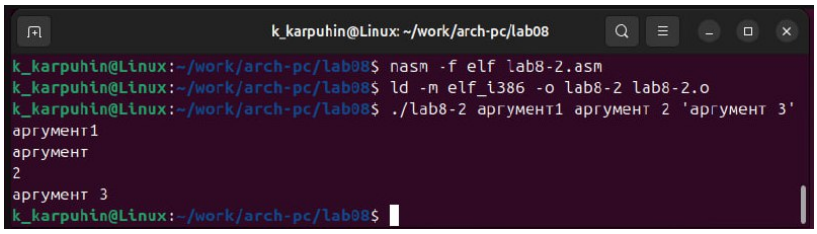

Рисунок 2.9: Запуск программы lab8-2 с несколькими аргументами командной строки и выводом каждого аргумента

Программой было обработано 4 аргумента, так как из-за отсутствия кавычек

выражение «аргумент 2» попадает в программу как два отдельных аргумента — «аргумент» и «2».

2.2.2 Программа вычисления суммы аргументов командной строки

Создал файл `lab8-3.asm` в каталоге `~/work/arch-pc/lab08` и ввёл в него текст программы из листинга 8.3, вычисляющей сумму числовых аргументов командной строки (рис. 2.10).

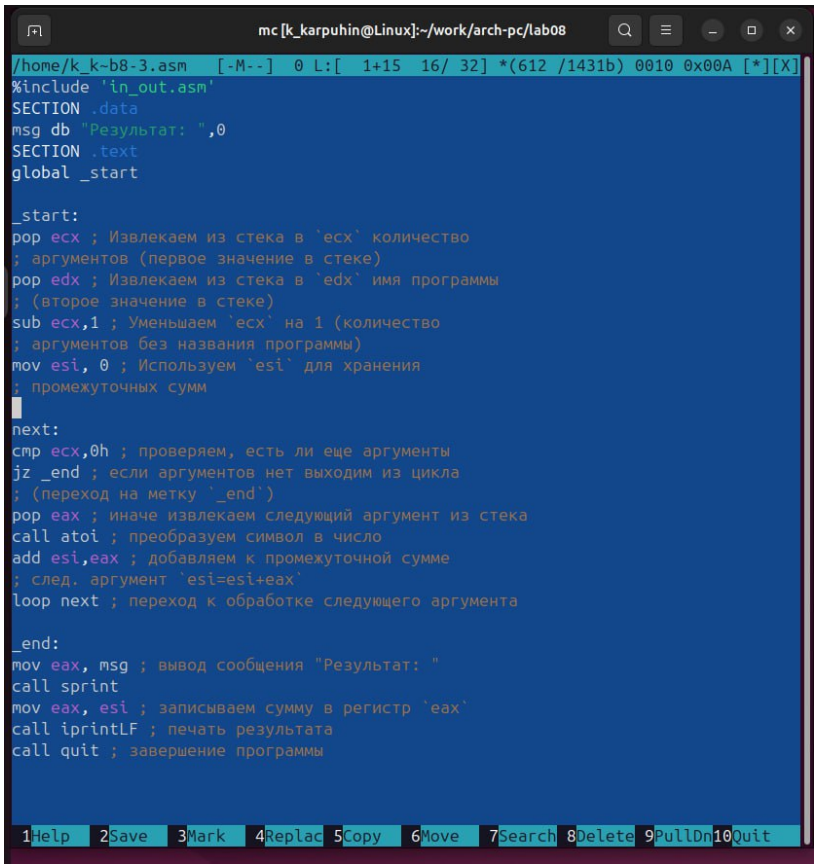

Рисунок 2.10: Текст программы lab8-3.asm, вычисляющей сумму аргументов командной строки (листинг 8.3)

Собрал и запустил программу, передав ей набор числовых аргументов:

```
./lab8-3 12 13 7 10 5
```

(рис. 2.11).

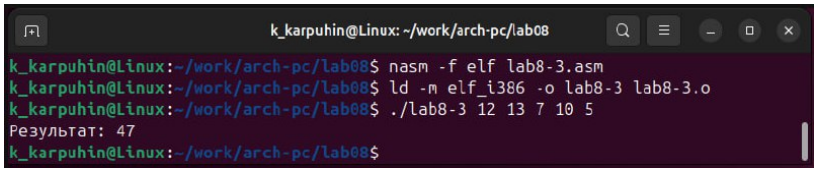

Рисунок 2.11: Создание и запуск программы lab8-3 с набором числовых аргументов, вывод суммы

2.2.3 Модификация программы для вычисления произведения аргументов

По заданию изменил программу из листинга 8.3 так, чтобы она вычисляла не сумму, а произведение аргументов командной строки.

Для этого внёс в текст программы следующие изменения:

1. Изменил начальное значение аккумулятора:

   вместо инициализации нулём

   ```
   mov esi, 0
   ```

   использовал единицу как нейтральный элемент умножения:

   ```
   mov esi, 1
   ```

2. Заменил операцию сложения на умножение:

   вместо

   ```
   add esi, eax
   ```

   использовал команду умножения:

```
imul esi, eax
```

(рис. 2.12).

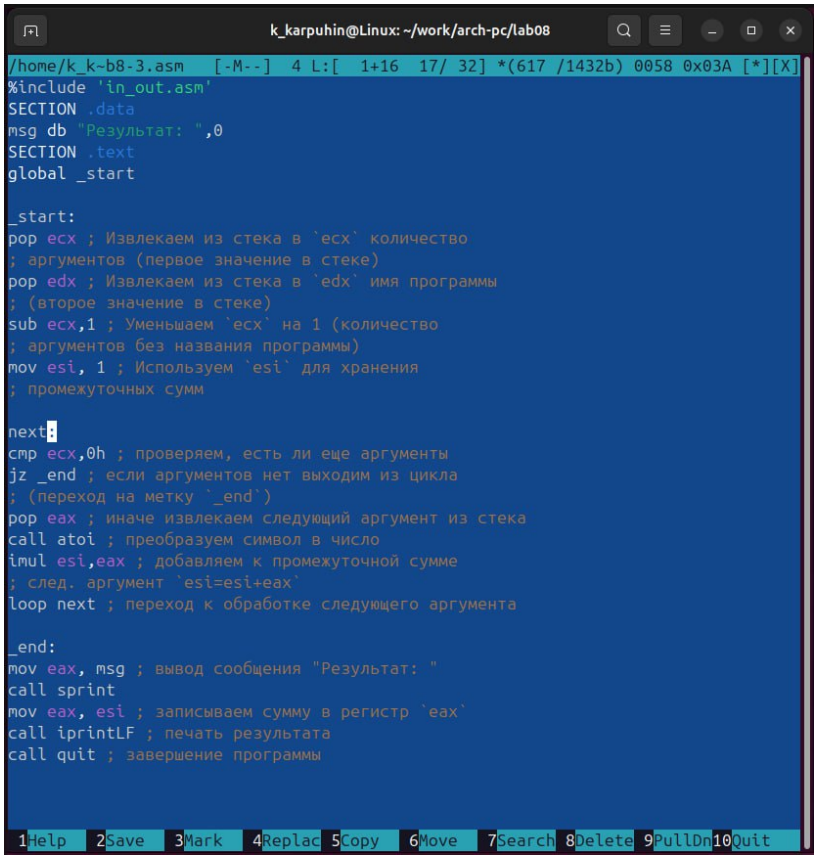

Рисунок 2.12: Изменённый текст программы lab8-3 для вычисления произведения аргументов

Собрал модифицированную программу и запустил её с набором числовых аргументов:

```
./lab8-3 6 7 2
```

(рис. 2.13).

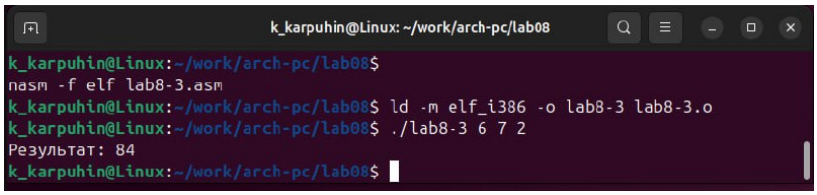

Рисунок 2.13: Создание и запуск модифицированной программы lab8-3, вычисляющей произведение аргументов

Программа корректно вычисляла произведение переданных аргументов:

- для набора 6 7 2 результатом было значение 84;
- при добавлении новых аргументов произведение изменялось соответственно.

Таким образом, на основе программы суммирования аргументов была реализована программа для вычисления произведения аргументов командной строки с использованием цикла и стека.

2.3 Задание для самостоятельной работы

2.3.1 Задание 1. Сумма значений функции f(x) для набора аргументов

В соответствии с пунктом 8.4 лабораторной работы необходимо написать программу, которая вычисляет сумму значений функции

$f(x)$

для набора аргументов

$x = x_1, x_2, \ldots, x_n,$

передаваемых через строку запуска программы.

Программа должна вычислять и выводить величину

$f(x_1) + f(x_2) + \ldots + f(x_n).$

Для своего варианта (вариант 1) использовал функцию $f(x)$ согласно таблице 8.1 методических указаний.

Создал файл с программой `lab8-task1.asm` и реализовал в нём следующий алгоритм:

1. Извлёк из стека количество аргументов и имя программы.
2. Уменьшил счётчик аргументов на 1, чтобы не учитывать имя программы.
3. Инициализировал регистр ESI (аккумулятор суммы) нулём.
4. В цикле:

- извлекал из стека очередной аргумент в виде строки;
- преобразовывал его в число с помощью процедуры atoi;
- вычислял значение f(x) = 2x + 15;
- добавлял полученное значение к сумме в регистре ESI. 5.После окончания цикла вывел на экран:
- строку с указанием вида функции f(x) = 2x + 15;
- строку с текстом «Результат:» и вычисленным значением суммы.

Текст программы, реализующей вычисление суммы значений функции `f(x)` для набора аргументов, приведён на рис. 2.14.

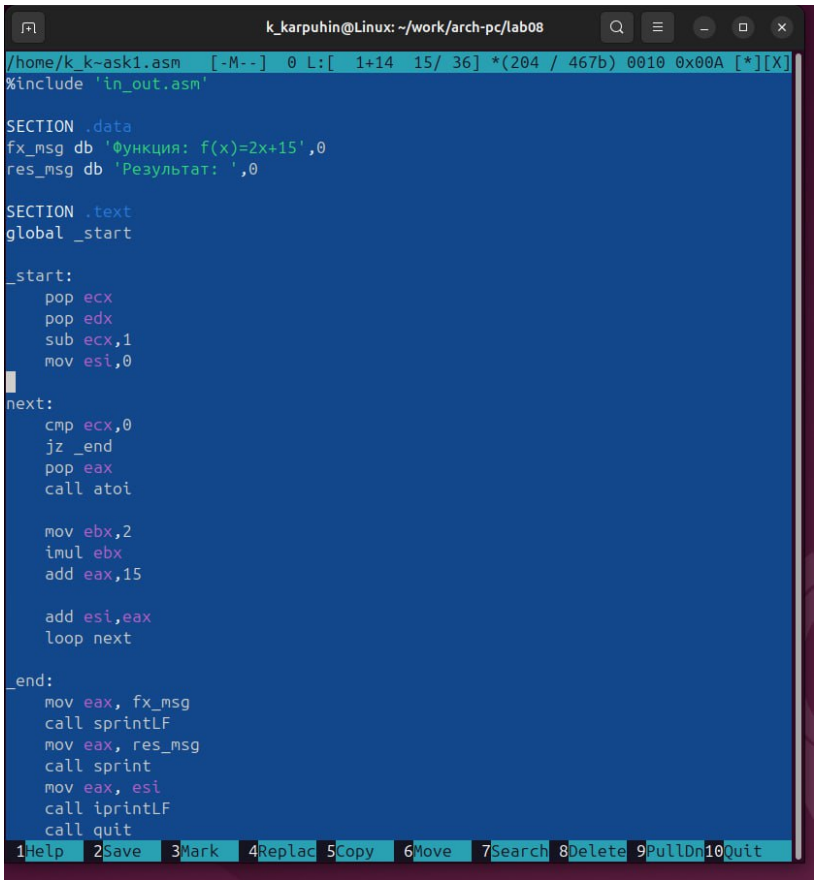

Рисунок 2.14: Текст программы, вычисляющей сумму значений функции f(x) для набора аргументов командной строки

Собрал и запустил программу с несколькими наборами аргументов x_1, x_2, . . . , x_n, проверяя корректность результата (рис. 2.15).

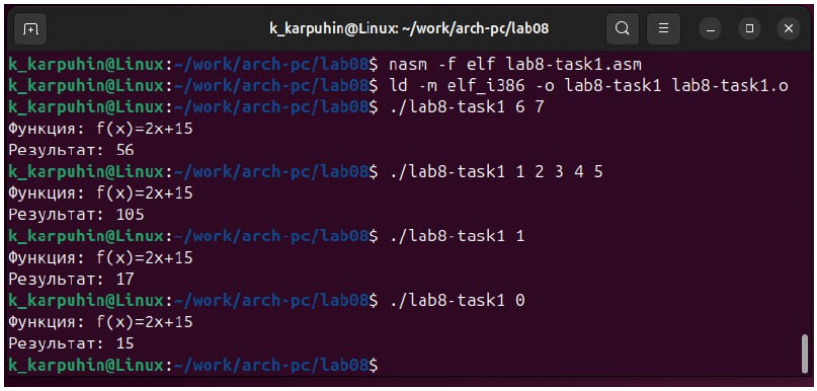

Рисунок 2.15: Запуск программы для нескольких наборов аргументов и проверка корректности результата

3 Выводы

В ходе выполнения данной лабораторной работы:

- изучил работу инструкции `loop` и особенности использования регистра `ecx` в качестве счётчика цикла;
- познакомился с организацией стека и на практике использовал инструкции `push` и `pop` для временного сохранения значений регистров внутри цикла;
- реализовал программу вывода значений регистра `ecx` и исследовал влияние дополнительных изменений счётчика на количество итераций цикла;
- освоил обработку аргументов командной строки в NASM, реализовав программы для вывода аргументов, вычисления их суммы и произведения;
- выполнил задание для самостоятельной работы, написав программу, которая вычисляет сумму значений функции `f(x)` для набора аргументов, передаваемых через командную строку.

В результате выполнения работы были закреплены навыки программирования циклов на языке ассемблера NASM, работы со стеком и обработки аргументов командной строки, что является важным шагом к пониманию низкоуровневой организации программ и взаимодействия с операционной системой.