РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ

по лабораторной работе № 8

дисциплина: Архитектура компьютера

Студент: Клименко Кирилл Русланович

Группа: НММбд-02-24

МОСКВА

2025г.

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

Приобретение навыков написания программ с использованием циклов и обработкой аргументов командной строки.

2 Задание

1. Создать каталог для программ лабораторной работы № 8, перейти в него и создать файл lab8-1.asm

2. Ввести в файл lab8-1.asm текст программы из листинга 1 методического указания. Создать исполняемый файл и запустить его. Посмотреть результаты работы.

3. Изменить текст программы, добавив изменение значения регистра ecx в цикле. Создать исполняемый файл и запустить его. Посмотреть результаты работы. Написать пояснение по результатам работы программы.

4. Внести изменения в текст программы, добавив команды push и pop (добавления в стек и извлечения из стека) для сохранения значения счетчика цикла loop . Создать исполняемый файл и проверить его работу. Написать пояснение по результатам работы программы.

5. Создать файл lab8-2.asm и ввести в него текст программы из листинга 2 методического указания. Создать исполняемый файл и запустить его, указав необходимые аргументы.

6. Создать файл lab8-3.asm и ввести в него текст программы из листинга 3 методического указания. Создать исполняемый файл и запустить его, указав необходимые аргументы. Написать пояснение по результатам работы программы.

7. Изменить текст Листинга 3 для вычисления произведения аргументов командной строки. Проверить результаты работы программы.

Задание для самостоятельной работы

1. Напишите программу, которая находит сумму значений функции f(x) для x= x1, x2,...xn, т.е. программа должна выводить значение f(x1)+f(x2)+...+f(xn). Значения xi передаются как аргументы. Вид функции f(x) выбрать из таблицы вариантов заданий в соответствии с вариантом, полученным при выполнении лабораторной работы № 6. Создайте исполняемый файл и проверьте его работу на нескольких наборах x=x1,x2, ...xn.

3 Теоретическое введение

Организация стека

Стек — это структура данных, организованная по принципу LIFO («Last In- First Out» или «последним пришёл — первым ушёл»). Стек является частью архитектуры процессора и реализован на аппаратном уровне. Для работы со стеком в процессоре есть специальные регистры (ss, bp, sp) и команды. Основной функцией стека является функция сохранения адресов возврата и передачи аргументов при вызове процедур. Кроме того, в нём выделяется память для локальных переменных и могут временно храниться значения регистров. Стек имеет вершину, адрес последнего добавленного элемента, который хранится в регистре esp (указатель стека). Противоположный конец стека называется дном. Значение, помещённое в стек последним, извлекается первым. При помещении значения в стек указатель стека уменьшается, а при извлечении — увеличивается. Для стека существует две основные операции:

-добавление элемента в вершину стека (push);

-извлечение элемента из вершины стека (pop).

Добавление элемента в стек

Команда push размещает значение в стеке, т.е. помещает значение в ячейку памяти, на которую указывает регистр esp, после этого значение регистра esp увеличивается на 4. Данная команда имеет один операнд — значение, которое необходимо поместить в стек. Существует ещё две команды для добавления значений в стек. Это команда pusha, которая помещает в стек содержимое всех регистров общего назначения в следующем порядке: ax, cx, dx, bx, sp, bp, si, di.

А также команда pushf, которая служит для перемещения в стек содержимого регистра флагов. Обе эти команды не имеют операндов.

Извлечение элемента из стека

Команда pop извлекает значение из стека, т.е. извлекает значение из ячейки памяти, на которую указывает регистр esp, после этого уменьшает значение регистра esp на 4. У этой команды также один операнд, который может быть регистром или переменной в памяти. Нужно помнить, что извлечённый из стека элемент не стирается из памяти и остаётся как “мусор”, который будет перезаписан при записи нового значения в стек Аналогично команде записи в стек существует команда popa, которая восстанавливает из стека все регистры общего назначения, и команда popf для перемещения значений из вершины стека в регистр флагов

Инструкции организации циклов

Для организации циклов существуют специальные инструкции. Для всех инструкций максимальное количество проходов задаётся в регистре ecx. Наиболее простой является инструкция loop. Она позволяет организовать безусловный цикл. Инструкция loop выполняется в два этапа. Сначала из регистра ecx вычитается единица и его значение сравнивается с нулём. Если регистр не равен нулю, то выполняется переход к указанной метке. Иначе переход не выполняется и управление передаётся команде, которая следует сразу после команды loop.

4 Выполнение лабораторной работы

1. Создали каталог для программ лабораторной работы № 8, перешли в него и создали файл lab8-1.asm (рис. 4.1 Создание каталога и файла для выполнения лабораторной работы).

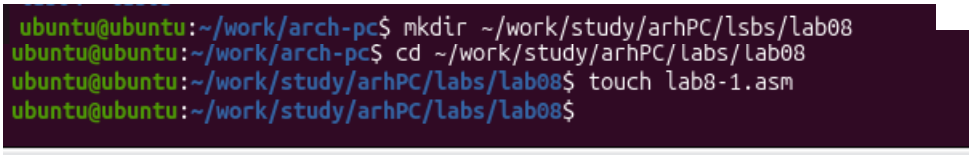


Рис. 4.1: Создание каталога и файла для выполнения лабораторной работы

2. Ввели в файл lab8-1.asm текст программы из листинга 1 (рис. 4.2 Листинг 1) методического указания. Создали исполняемый файл и запустили его. Посмотрели результаты работы (рис. 4.3 Результаты работы программы из листинга 1 методического указания).

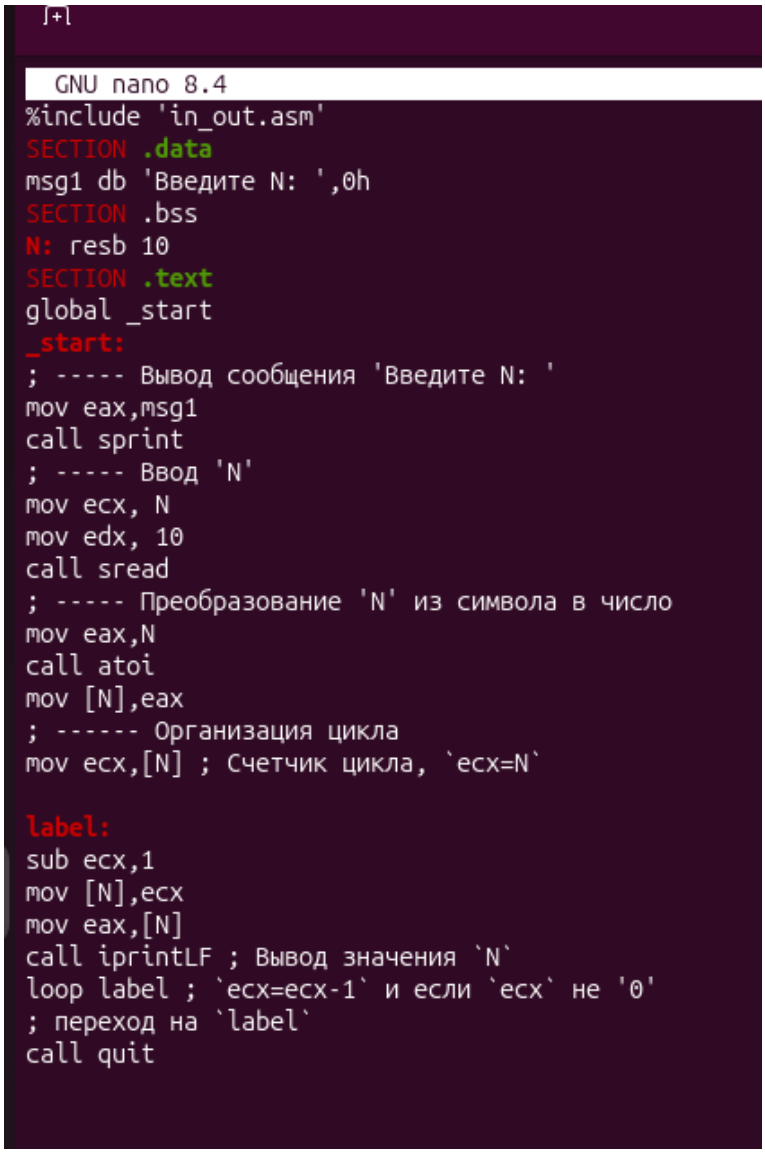
```
 GNU nano 8.4                                                  /home/ub
%include 'in_out.asm'
SECTION .data
msg1 db 'Введите N: ',0h
SECTION .bss
N: resb 10
SECTION .text
global _start
_start:
; ----- Вывод сообщения 'Введите N: '
mov eax,msg1
call sprint
; ----- Ввод 'N'
mov ecx, N
mov edx, 10
call sread
; ----- Преобразование 'N' из символа в число
mov eax,N
call atoi
mov [N],eax
; ------ Организация цикла
mov ecx,[N] ; Счетчик цикла, `ecx=N`
label:
mov [N],ecx
mov eax,[N]
call iprintLF ; Вывод значения `N`
loop label ; `ecx=ecx-1` и если `ecx` не '0'
; переход на `label`
call quit
```

Рис. 4.2: Листинг 1

```
ubuntu@ubuntu:~/work/study/arhPC/labs$ cp ~/work/study/arhPC/labs/lab05/in_out.asm ~/work/study/arhPC/labs/lab08/
ubuntu@ubuntu:~/work/study/arhPC/labs$ cd ~/work/study/arhPC/labs/lab08
ubuntu@ubuntu:~/work/study/arhPC/labs/lab08$ ls
in_out.asm  lab8-1.asm  list  presentation  report
ubuntu@ubuntu:~/work/study/arhPC/labs/lab08$ nasm -f elf lab8-1.asm
ubuntu@ubuntu:~/work/study/arhPC/labs/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
ubuntu@ubuntu:~/work/study/arhPC/labs/lab08$ ./lab8-1
Введите N: 5
5
4
3
2
1
ubuntu@ubuntu:~/work/study/arhPC/labs/lab08$
```

Рис. 4.3: Результаты работы программы из листинга 1 методического указания

3. Изменили текст программы добавив изменение значение регистра ecx в цикле (рис.4.4). Посмотрели результаты работы (рис. 4.5).



```
 ]+l

  GNU nano 8.4
%include 'in_out.asm'
SECTION .data
msg1 db 'Введите N: ',0h
SECTION .bss
N: resb 10
SECTION .text
global _start
_start:
; ----- Вывод сообщения 'Введите N: '
mov eax,msg1
call sprint
; ----- Ввод 'N'
mov ecx, N
mov edx, 10
call sread
; ----- Преобразование 'N' из символа в число
mov eax,N
call atoi
mov [N],eax
; ------ Организация цикла
mov ecx,[N] ; Счетчик цикла, `ecx=N`

label:
sub ecx,1
mov [N],ecx
mov eax,[N]
call iprintLF ; Вывод значения `N`
loop label ; `ecx=ecx-1` и если `ecx` не '0'
; переход на `label`
call quit
```

Рис. 4.4: Листинг 1 с внесенными изменениями согласно методическому указанию

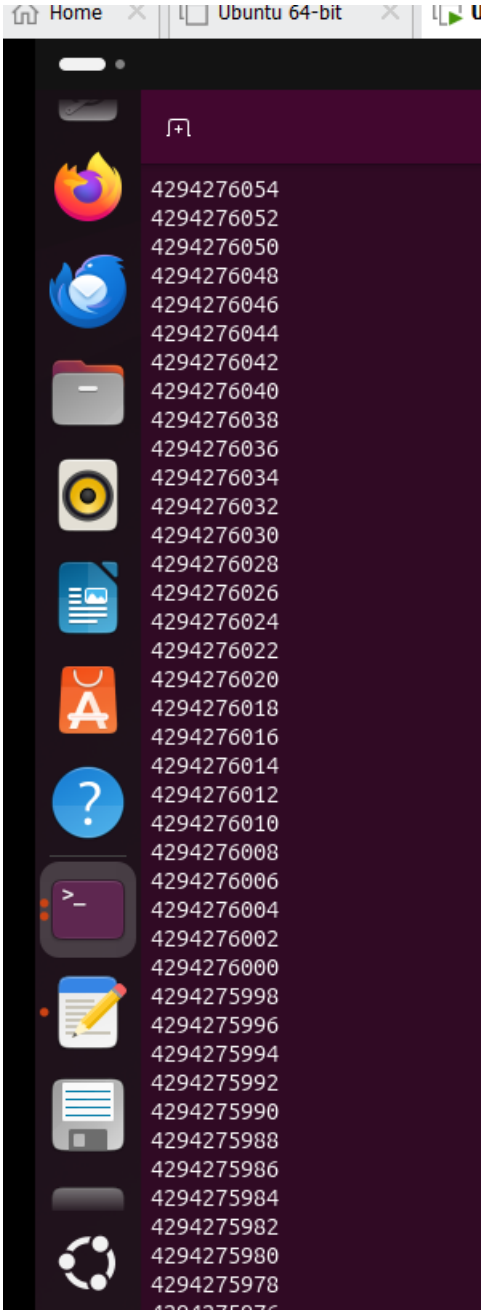


Рис. 4.5: Результаты работы программы из Листинга 1 с внесенными изменениями согласно методическому указанию

Цикл стал бесконечным. Образовалось кольцо.

4. Для использования регистра ecx в цикле и сохранения корректности работы программы можно использовать стек. Внесли изменения в текст программы добавив команды push и pop (добавления в стек и извлечения из стека) для

сохранения значения счетчика цикла loop (рис. 4.6). Создали исполняемый файл и проверили его работу (рис. 4.7).

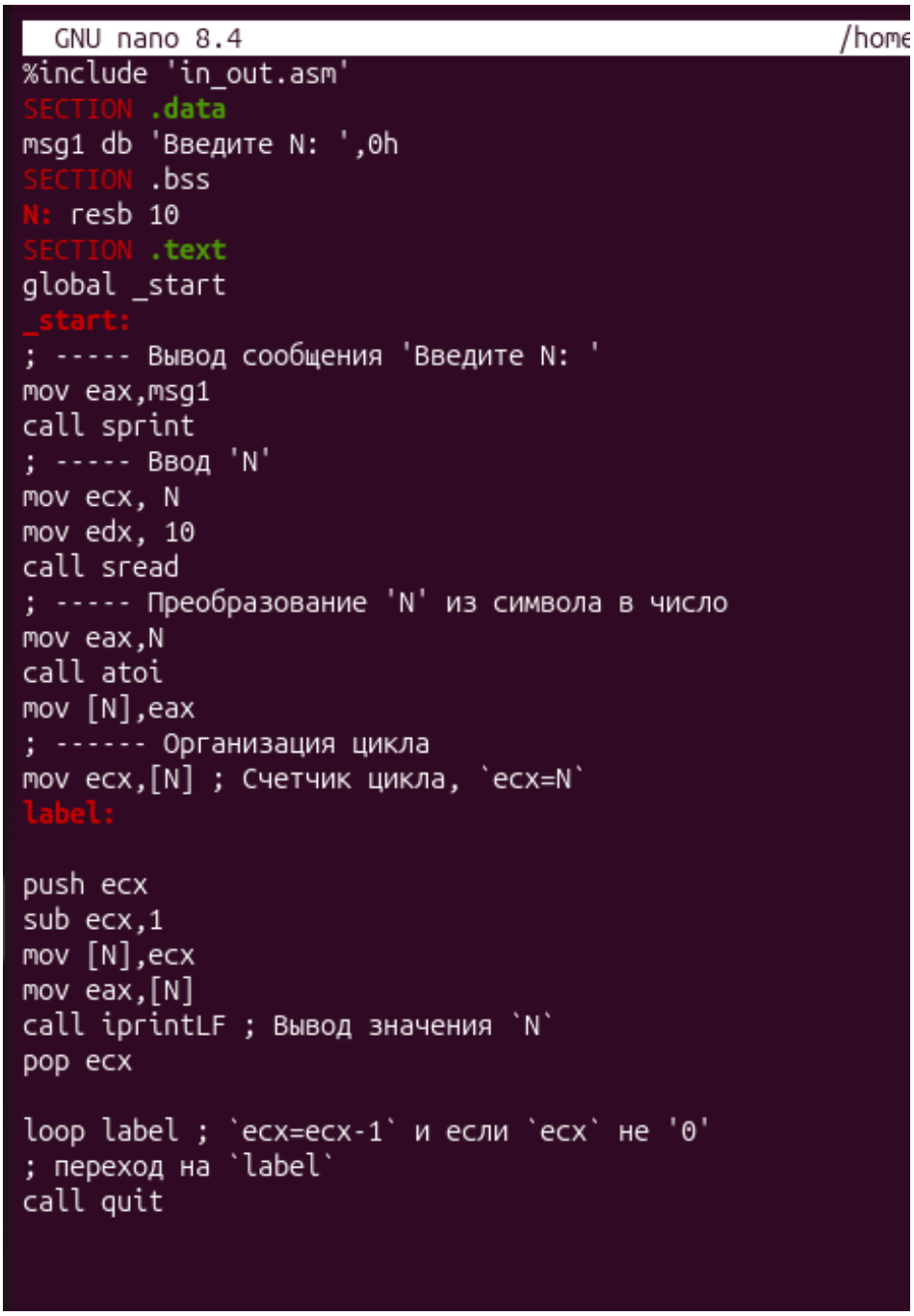


Рис. 4.6: Измененный Листинг 1 с внесенными командами push и pop

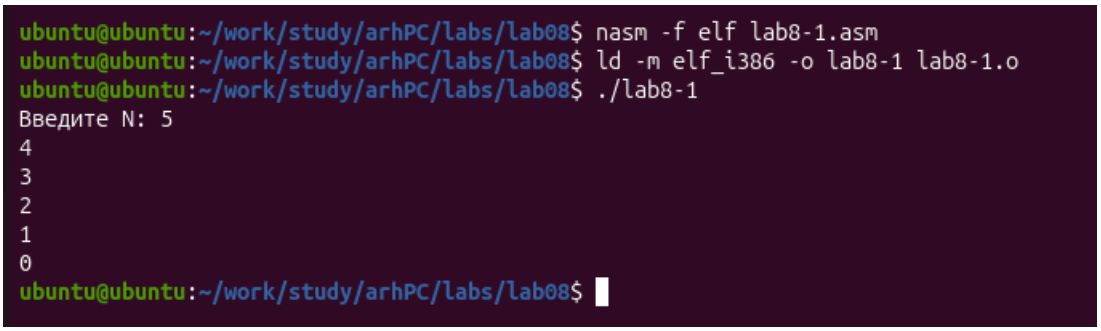


Рис. 4.7: Результаты работы программы измененного Листинга 1 с внесенными командами push и pop

Цикл и счетчик отработал правильно. По итогу после изменения программы, число проходки циклов стал соответствовать числу введенному с клавиатуры.

5. Создали файл lab8-2.asm и ввели в него текст программы из листинга 2 методического указания (рис. 4.8). Создали исполняемый файл и запустили его, указав необходимые аргументы. Программа выводит все 3 аргумента, которые ввели, но в разной вариации (рис. 4.9).

```
 GNU nano 8.4                                          /home/ub
%include 'in_out.asm'
SECTION .text
global _start
_start:
pop ecx ; Извлекаем из стека в `ecx` количество
; аргументов (первое значение в стеке)
pop edx ; Извлекаем из стека в `edx` имя программы
; (второе значение в стеке)
sub ecx, 1 ; Уменьшаем `ecx` на 1 (количество
; аргументов без названия программы)
next:
cmp ecx, 0 ; проверяем, есть ли еще аргументы
jz _end ; если аргументов нет выходим из цикла
; (переход на метку `_end`)
pop eax ; иначе извлекаем аргумент из стека
call sprintLF ; вызываем функцию печати
loop next ; переход к обработке следующего
; аргумента (переход на метку `next`)
_end:
call quit
```

Рис. 4.8: Листинг 2

```
ubuntu@ubuntu:~/work/study/arhPC/labs/lab08$ nano list2
ubuntu@ubuntu:~/work/study/arhPC/labs/lab08$ touch lab8-2.asm
ubuntu@ubuntu:~/work/study/arhPC/labs/lab08$ mc

ubuntu@ubuntu:~/work/study/arhPC/labs/lab08$ nasm -f elf lab8-2.asm
ubuntu@ubuntu:~/work/study/arhPC/labs/lab08$ ld -m elf_i386 -o lab8-2 lab8-2.o
ubuntu@ubuntu:~/work/study/arhPC/labs/lab08$ ./lab8-2
ubuntu@ubuntu:~/work/study/arhPC/labs/lab08$ ./lab8-2 аргумент1 аргумент 2 "аргумент 3"
аргумент1
аргумент
2
аргумент 3
ubuntu@ubuntu:~/work/study/arhPC/labs/lab08$
```

Рис. 4.9: Результат работы программы.

6. Создали файл lab8-3.asm и ввели в него текст программы из листинга 3 методического указания (рис. 4.10). Создали исполняемый файл и запустили его, указав необходимые аргументы. Программа вывела сумму

чисел, которые мы ввели (рис. 4.11).

```
 GNU nano 8.4                                          /home/ubu
%include 'in_out.asm'
SECTION .data
msg db "Результат: ",0
SECTION .text
global _start
_start:
pop ecx ; Извлекаем из стека в `ecx` количество
; аргументов (первое значение в стеке)
pop edx ; Извлекаем из стека в `edx` имя программы
; (второе значение в стеке)
sub ecx,1 ; Уменьшаем `ecx` на 1 (количество
; аргументов без названия программы)
mov esi, 0 ; Используем `esi` для хранения
; промежуточных сумм
next:
cmp ecx,0h ; проверяем, есть ли еще аргументы
jz _end ; если аргументов нет выходим из цикла
; (переход на метку `_end`)
pop eax ; иначе извлекаем следующий аргумент из стека
call atoi ; преобразуем символ в число
add esi,eax ; добавляем к промежуточной сумме
; след. аргумент `esi=esi+eax`
loop next ; переход к обработке следующего аргумента
_end:
mov eax, msg ; вывод сообщения "Результат: "
call sprint
mov eax, esi ; записываем сумму в регистр `eax`
call iprintLF ; печать результата
call quit ; завершение программы
```

Рис. 4.10: Листинг 3

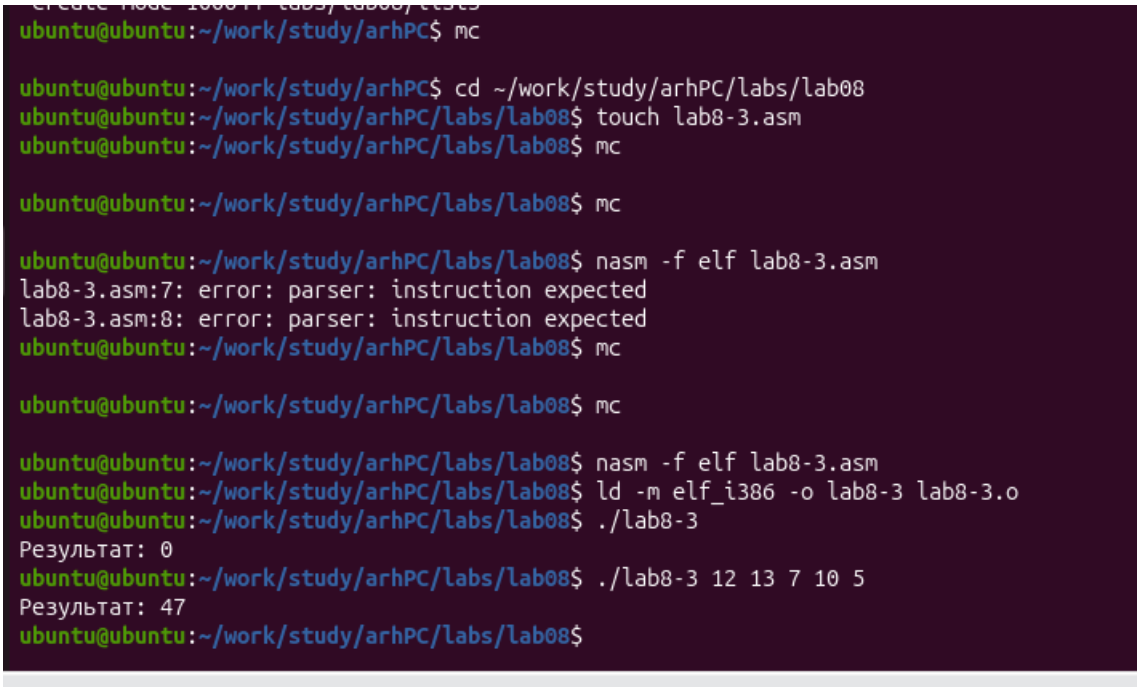


Рис. 4.11: Результат работы программы из Листинга 3

Изменили текст Листинга 3 для вычисления произведения аргументов командной строки (рис. 4.12). Проверили результаты работы программы (рис. 4.13).

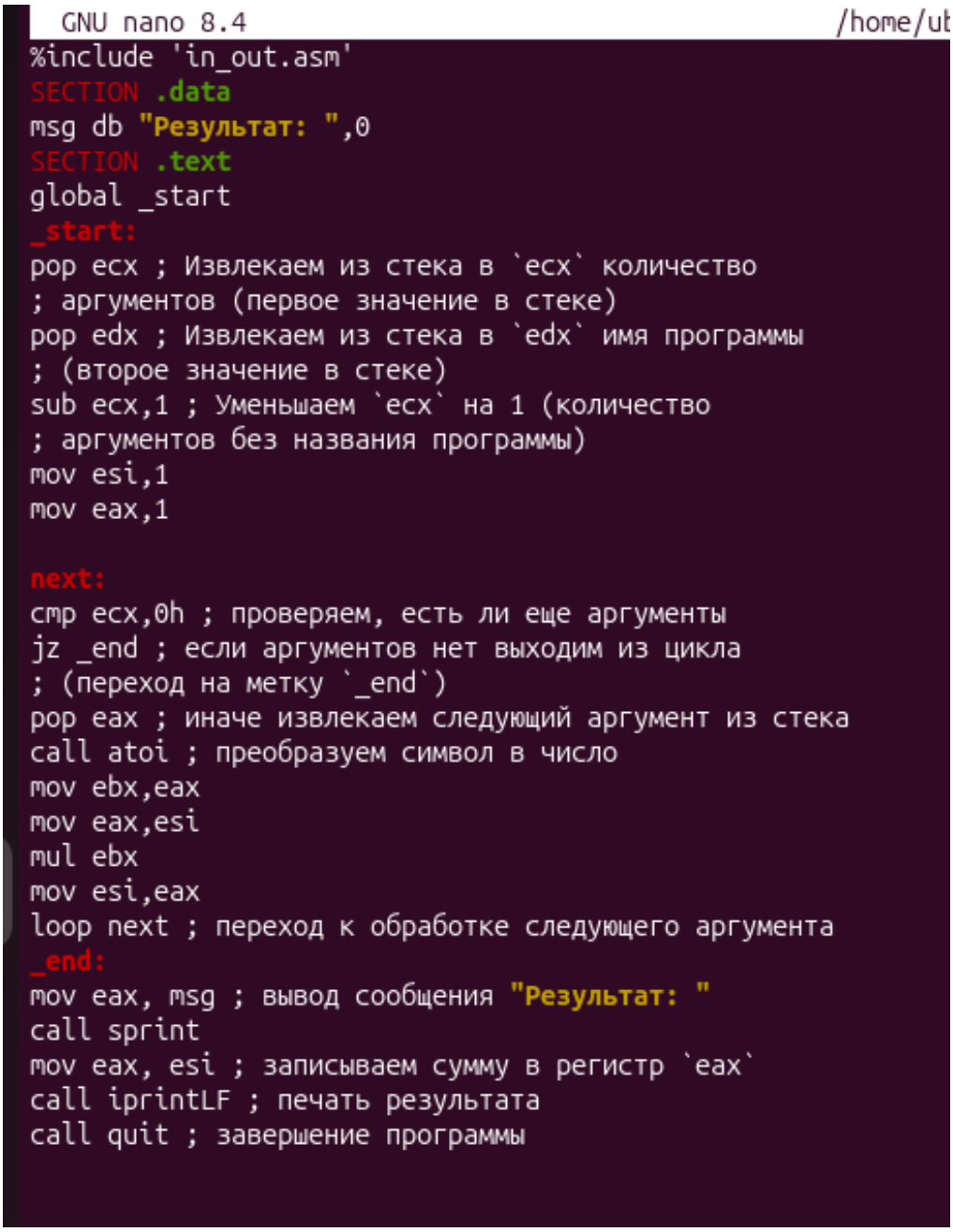


Рис. 4.12: Измененный Листинг 3 для вычисления произведения аргументов командной строки

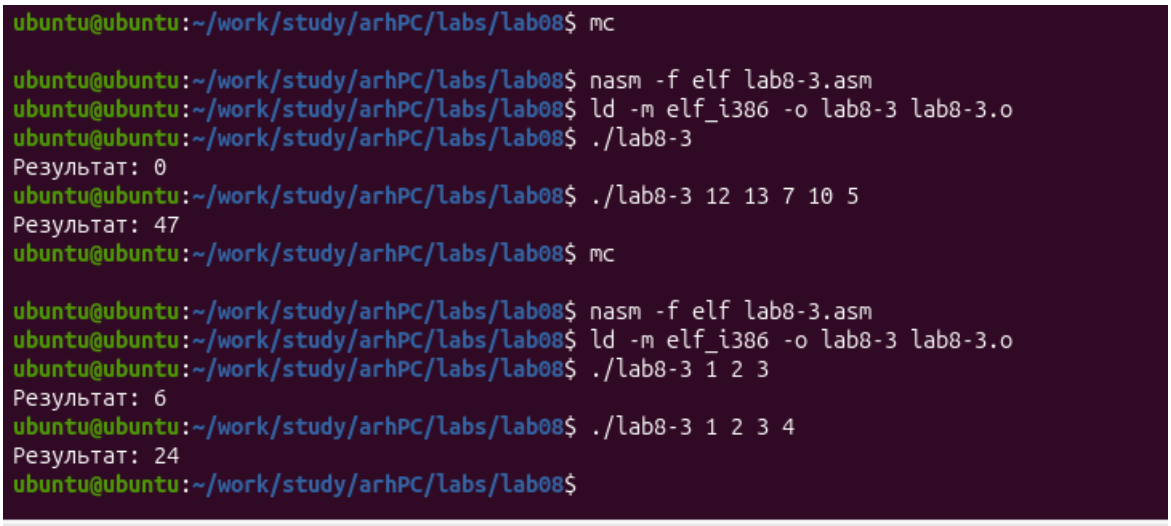


Рис. 4.13: Результаты работы программы для вычисления произведения аргументов командной строки

Задание для самостоятельной работы

1. Напишите программу, которая находит сумму значений функции f(x) для x= x1, x2,...xn, т.е. программа должна выводить значение f(x1)+f(x2)+...+f(xn). Значения xi передаются как аргументы. Вид функции f(x) выбрать из таблицы вариантов заданий в соответствии с вариантом, полученным при выполнении лабораторной работы № 7. Создайте исполняемый файл и проверьте его работу на нескольких наборах x=x1,x2, ...xn.

При выполнении лабораторной работы №7 у меня получился вариант 8, соответственно для варианта №8 f(x) = 7 + 2x

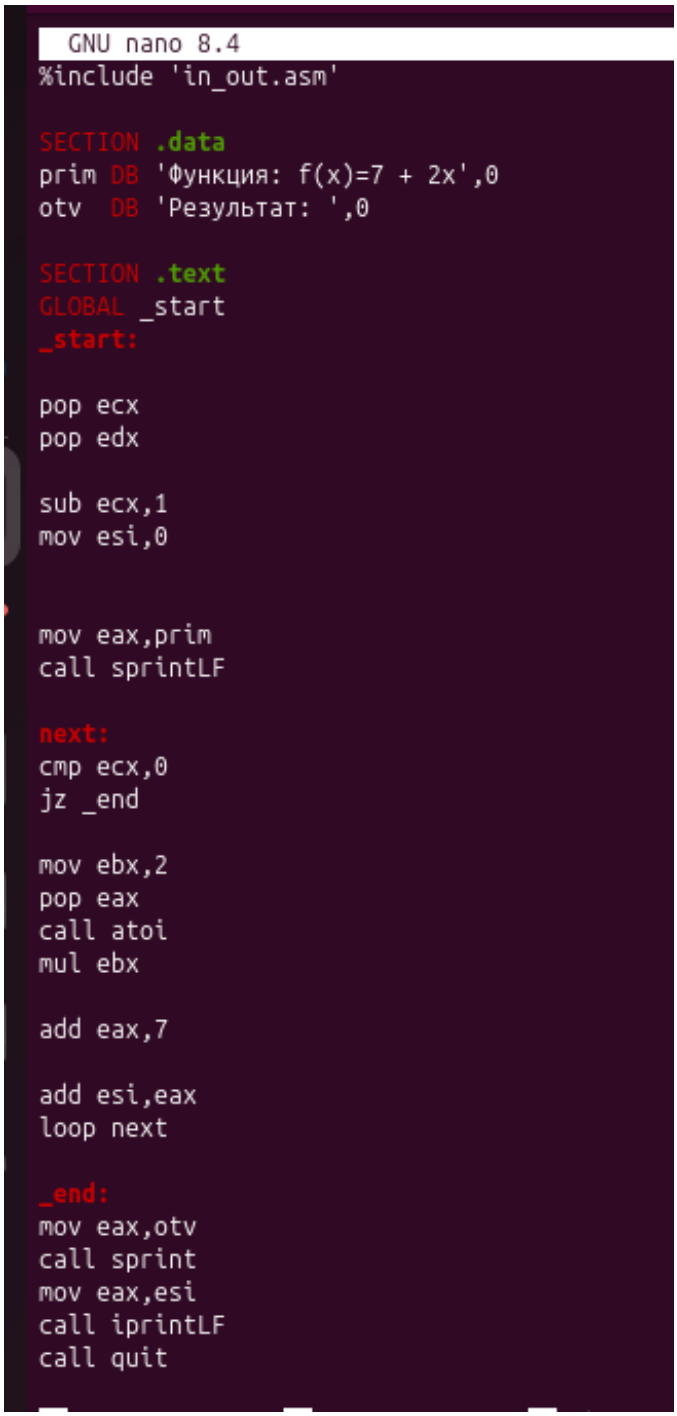


Рис. 4.14: Листинг самостоятельного задания №1

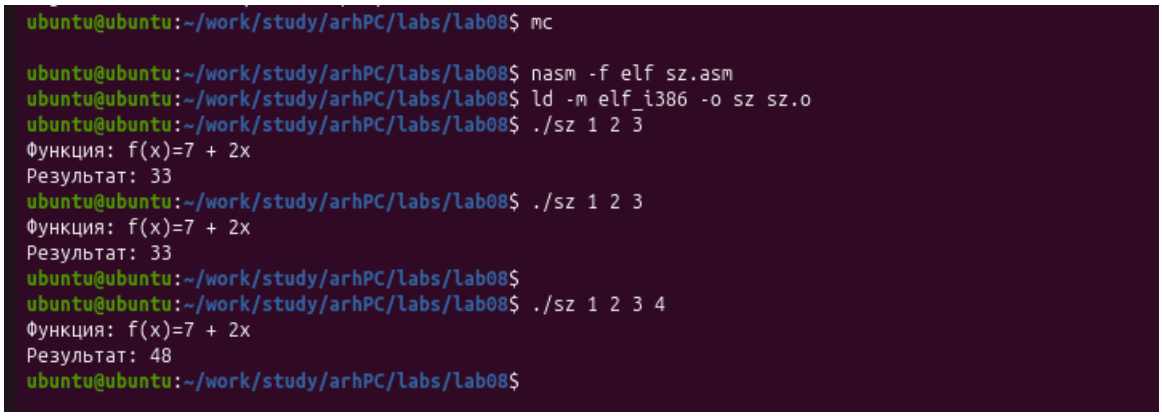


Рис. 4.15: Результаты работы программы по самостоятельному заданию №1

5 Выводы

Приобрели навыки написания программ с использованием циклов и обработкой аргументов командной строки.

Список литературы

1. GDB: The GNU Project Debugger. — URL:https://www.gnu.org/software/gdb/.
2. GNU Bash Manual. — 2016. — URL: https://www.gnu.org/software/bash/manual/.
3. Midnight Commander Development Center. — 2021. — URL: https://midnight-commander. org/.
4. NASM Assembly Language Tutorials. — 2021. — URL: https://asmtutor.com/.
5. Newham C. Learning the bash Shell: Unix Shell Programming. — O'Reilly Media, 2005. — 354 c. — (In a Nutshell). — ISBN 0596009658. — URL: http://www.amazon.com/Learningbash-Shell-Programming-Nutshell/dp/0596009658.
6. Robbins A. Bash Pocket Reference. — O'Reilly Media, 2016. — 156 c. — ISBN 978-1491941591.
7. The NASM documentation. — 2021. — URL: https://www.nasm.us/docs.php.
8. Zarrelli G. Mastering Bash. — Packt Publishing, 2017. — 502 c. — ISBN 9781784396879.
9. Колдаев В. Д., Лупин С. А. Архитектура ЭВМ. — М. : Форум, 2018.
10. Куляс О. Л., Никитин К. А. Курс программирования на ASSEMBLER. — М. : Солон-Пресс, 2017.
11. Новожилов О. П. Архитектура ЭВМ и систем. — М. : Юрайт, 2016.
12. Расширенный ассемблер: NASM. — 2021. — URL: https://www.opennet.ru/docs/RUS/nasm/.
13. Робачевский А., Немнюгин С., Стесик О. Операционная система UNIX. — 2-е изд. — БХВПетербург, 2010. — 656 с. — ISBN 978-5-94157-538-1.
14. Столяров А. Программирование на языке ассемблера NASM для ОС Unix. — 2-е изд. — М. : МАКС Пресс, 2011. — URL: http://www.stolyarov.info/books/asm_unix.

15. Таненбаум Э. Архитектура компьютера. — 6-е изд. — СПб. : Питер, 2013. — 874 с. — (Классика Computer Science).
16. Таненбаум Э., Бос Х. Современные операционные системы. — 4-е изд. — СПб. : Питер, 2015. — 1120 с. — (Классика Computer Science).