

# **Отчет по лабораторной работе №6**

**Архитектура компьютера и операционные системы**

Александр Дмитриевич Собко

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Задание</b>	<b>6</b>
<b>3</b>	<b>Теоретическое введение</b>	<b>7</b>
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>9</b>
4.1	Реализация циклов в NASM . . . . .	9
4.2	Обработка аргументов командной строки . . . . .	13
<b>5</b>	<b>Задание для самостоятельной работы</b>	<b>16</b>
<b>6</b>	<b>Выводы</b>	<b>18</b>
	<b>Список литературы</b>	<b>19</b>

## Список иллюстраций

4.1	Рисунок1	10
4.2	Рисунок2	10
4.3	Рисунок3	11
4.4	Рисунок4	11
4.5	Рисунок5	12
4.6	Рисунок6	12
4.7	Рисунок7	13
4.8	Рисунок8	14
4.9	Рисунок9	14
4.10	Рисунок10	15
4.11	Рисунок11	15
4.12	Рисунок12	15
5.1	Рисунок13	17
5.2	Рисунок14	17

## Список таблиц

# 1 Цель работы

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

## 2 Задание

Отредактировать данные программы с циклами и использованием аргументов командной строки, понять как работают в ассемблере циклы и что самое главное СТЕК.

### 3 Теоретическое введение

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. На рис. 8.1 показана схема организации стека в процессоре. 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).

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.

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.

Для организации циклов существуют специальные инструкции. Для всех инструкций максимальное количество проходов задаётся в регистре `ecx`. Наиболее простой является инструкция `loop`.



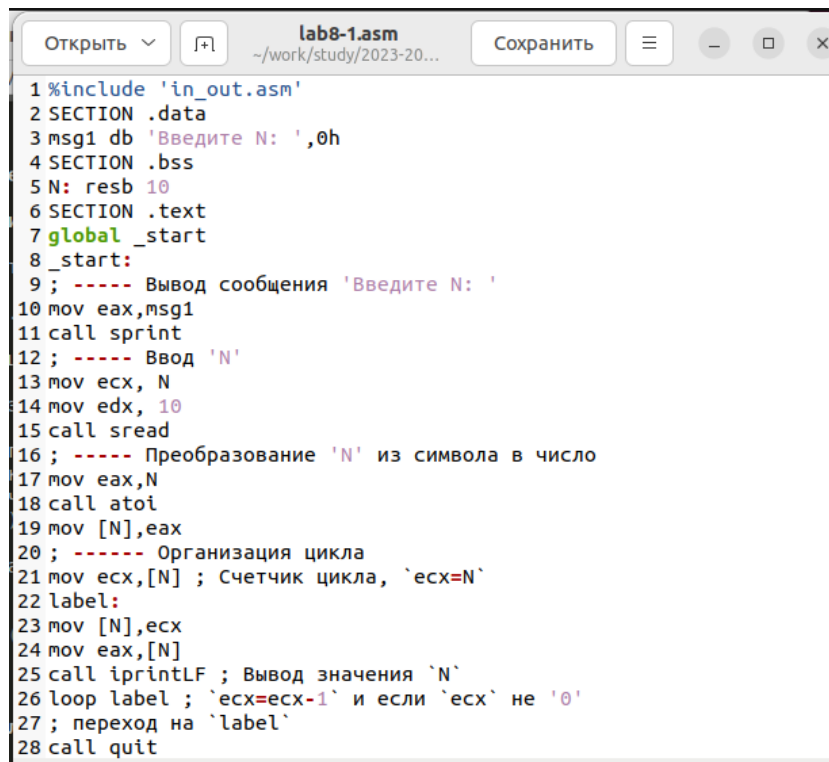
## 4 Выполнение лабораторной работы

### 4.1 Реализация циклов в NASM

Создайте каталог для программ лабораторной работы № 8, перейдите в него и создайте файл lab8-1.asm

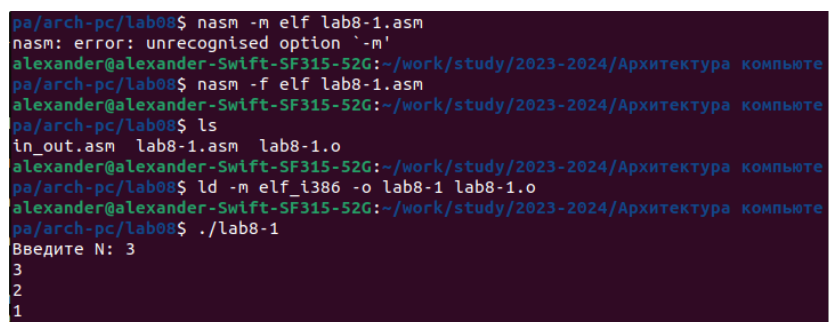
При реализации циклов в NASM с использованием инструкции `loop` необходимо помнить о том, что эта инструкция использует регистр `ecx` в качестве счетчика и на каждом шаге уменьшает его значение на единицу. В качестве примера рассмотрим программу, которая выводит значение регистра `ecx`. Внимательно изучите текст программы (Листинг 8.1).

Введите в файл lab8-1.asm текст программы из листинга 8.1. Создайте исполняемый файл и проверьте его работу.



```
1 %include 'in_out.asm'
2 SECTION .data
3 msg1 db 'Введите N: ',0h
4 SECTION .bss
5 N: resb 10
6 SECTION .text
7 global _start
8 _start:
9 ; ----- Вывод сообщения 'Введите N: '
10 mov eax,msg1
11 call sprint
12 ; ----- Ввод 'N'
13 mov ecx, N
14 mov edx, 10
15 call sread
16 ; ----- Преобразование 'N' из символа в число
17 mov eax,N
18 call atoi
19 mov [N],eax
20 ; ----- Организация цикла
21 mov ecx,[N] ; Счетчик цикла, `ecx=N`
22 label:
23 mov [N],ecx
24 mov eax,[N]
25 call iprintLF ; Вывод значения `N`
26 loop label ; `ecx=ecx-1` и если `ecx` не `0`
27 ; переход на `label`
28 call quit
29
```

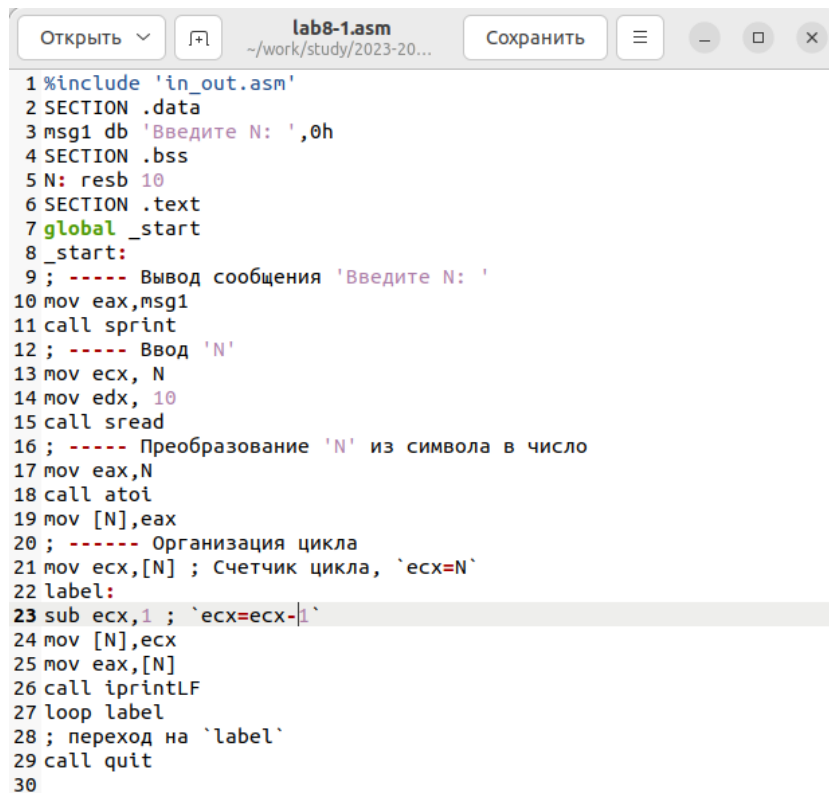
Рис. 4.1: Рисунок1



```
pa/arch-pc/lab08$ nasm -m elf lab8-1.asm
nasm: error: unrecognized option `-m'
alexander@alexander-Swift-SF315-52G:~/work/study/2023-2024/Архитектура компьюте
pa/arch-pc/lab08$ nasm -f elf lab8-1.asm
alexander@alexander-Swift-SF315-52G:~/work/study/2023-2024/Архитектура компьюте
pa/arch-pc/lab08$ ls
in_out.asm lab8-1.asm lab8-1.o
alexander@alexander-Swift-SF315-52G:~/work/study/2023-2024/Архитектура компьюте
pa/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
alexander@alexander-Swift-SF315-52G:~/work/study/2023-2024/Архитектура компьюте
pa/arch-pc/lab08$ ./lab8-1
Введите N: 3
3
2
1
```

Рис. 4.2: Рисунок2

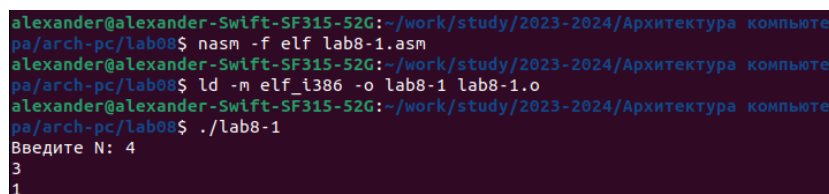
Данный пример показывает, что использование регистра ecx в теле цикла loop может привести к некорректной работе программы. Измените текст программы добавив изменение значение регистра ecx в цикле



```
1 %include 'in_out.asm'
2 SECTION .data
3 msg1 db 'Введите N: ',0h
4 SECTION .bss
5 N: resb 10
6 SECTION .text
7 global _start
8 _start:
9 ; ----- Вывод сообщения 'Введите N: '
10 mov eax,msg1
11 call sprint
12 ; ----- Ввод 'N'
13 mov ecx, N
14 mov edx, 10
15 call sread
16 ; ----- Преобразование 'N' из символа в число
17 mov eax,N
18 call atoi
19 mov [N],eax
20 ; ----- Организация цикла
21 mov ecx,[N] ; Счетчик цикла, `ecx=N`
22 label:
23 sub ecx,1 ; `ecx=ecx-1`
24 mov [N],ecx
25 mov eax,[N]
26 call iprintLF
27 loop label
28 ; переход на `label`
29 call quit
30
```

Рис. 4.3: Рисунок3

Создайте исполняемый файл и проверьте его работу. Какие значения принимает регистр `ecx` в цикле? Соответствует ли число проходов цикла значению `N` введенному с клавиатуры?

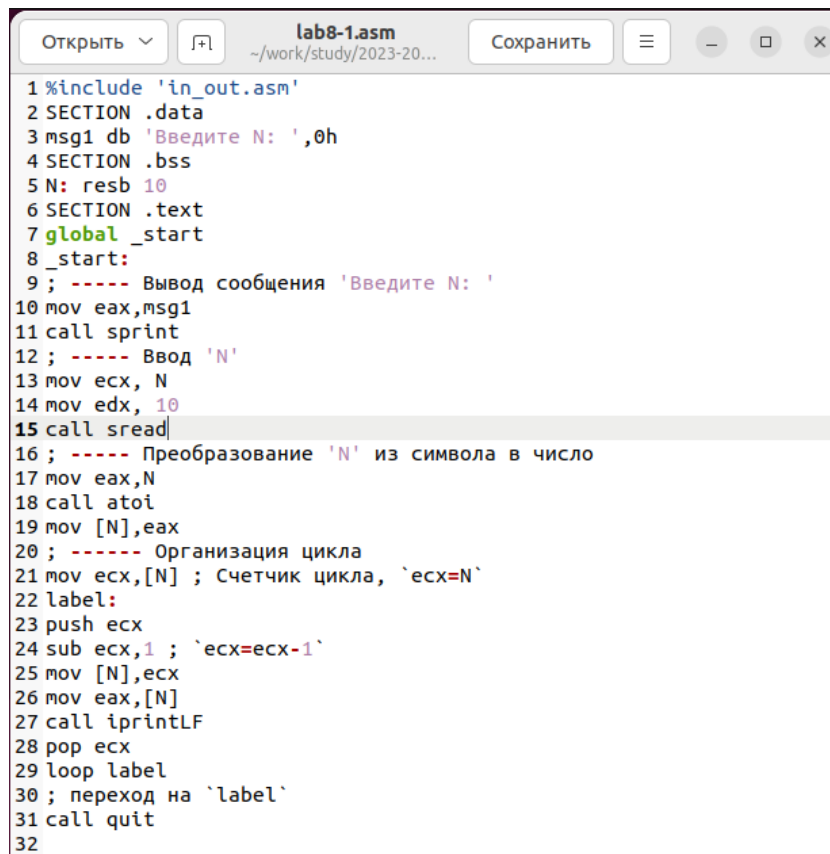


```
alexander@alexander-Swift-SF315-52G:~/work/study/2023-2024/Архитектура компьютера
pa/arch-pc/lab08$ nasm -f elf lab8-1.asm
alexander@alexander-Swift-SF315-52G:~/work/study/2023-2024/Архитектура компьютера
pa/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
alexander@alexander-Swift-SF315-52G:~/work/study/2023-2024/Архитектура компьютера
pa/arch-pc/lab08$ ./lab8-1
Введите N: 4
3
1
```

Рис. 4.4: Рисунок4

Принимает нечетные значения. Не соответствует.

Для использования регистра `ecx` в цикле и сохранения корректности работы программы можно использовать стек. Внесите изменения в текст программы добавив команды `push` и `pop` (добавления в стек и извлечения из стека) для сохранения значения счетчика цикла `loop`



```
1 %include 'in_out.asm'
2 SECTION .data
3 msg1 db 'Введите N: ',0h
4 SECTION .bss
5 N: resb 10
6 SECTION .text
7 global _start
8 _start:
9 ; ----- Вывод сообщения 'Введите N: '
10 mov eax,msg1
11 call sprint
12 ; ----- Ввод 'N'
13 mov ecx, N
14 mov edx, 10
15 call sread
16 ; ----- Преобразование 'N' из символа в число
17 mov eax,N
18 call atoi
19 mov [N],eax
20 ; ----- Организация цикла
21 mov ecx,[N] ; Счетчик цикла, `ecx=N`
22 label:
23 push ecx
24 sub ecx,1 ; `ecx=ecx-1`
25 mov [N],ecx
26 mov eax,[N]
27 call iprintLF
28 pop ecx
29 loop label
30 ; переход на `label`
31 call quit
32
```

Рис. 4.5: Рисунок5

Создайте исполняемый файл и проверьте его работу. Соответствует ли в данном случае число проходов цикла значению N введенному с клавиатуры?



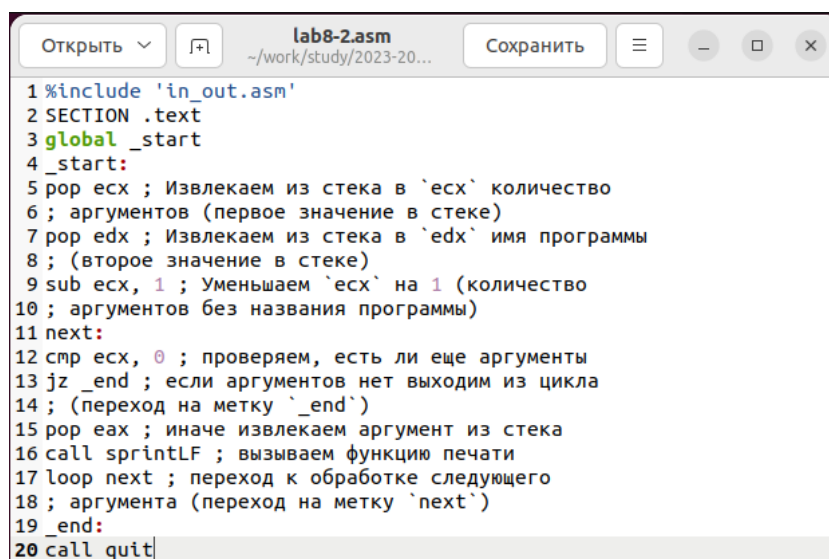
```
alexander@alexander-Swift-SF315-S2G: ~/work/study/2023-2024/Архитектура компьютера
pa/arch-pc/lab08$ nasm -f elf lab8-1.asm
alexander@alexander-Swift-SF315-S2G: ~/work/study/2023-2024/Архитектура компьютера
pa/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
alexander@alexander-Swift-SF315-S2G: ~/work/study/2023-2024/Архитектура компьютера
pa/arch-pc/lab08$ ./lab8-1
Введите N: 5
4
3
2
1
0
alexander@alexander-Swift-SF315-S2G: ~/work/study/2023-2024/Архитектура компьютера
pa/arch-pc/lab08$ ./lab8-1
Введите N: 4
3
2
1
0
alexander@alexander-Swift-SF315-S2G: ~/work/study/2023-2024/Архитектура компьютера
```

Рис. 4.6: Рисунок6

Соответствует.

## 4.2 Обработка аргументов командной строки

При разработке программ иногда встает необходимость указывать аргументы, которые будут использоваться в программе, непосредственно из командной строки при запуске программы. При запуске программы в NASM аргументы командной строки загружаются в стек в обратном порядке, кроме того в стек записывается имя программы и общее количество аргументов. Последние два элемента стека для программы, скомпилированной NASM, – это всегда имя программы и количество переданных аргументов. Таким образом, для того чтобы использовать аргументы в программе, их просто нужно извлечь из стека. Обработку аргументов нужно проводить в цикле. Т.е. сначала нужно извлечь из стека количество аргументов, а затем циклично для каждого аргумента выполнить логику программы. В качестве примера рассмотрим программу, которая выводит на экран аргументы командной строки. Внимательно изучите текст программы (Листинг 8.2).



```
1 %include 'in_out.asm'
2 SECTION .text
3 global _start
4 _start:
5 pop ecx ; Извлекаем из стека в `ecx` количество
6 ; аргументов (первое значение в стеке)
7 pop edx ; Извлекаем из стека в `edx` имя программы
8 ; (второе значение в стеке)
9 sub ecx, 1 ; Уменьшаем `ecx` на 1 (количество
10 ; аргументов без названия программы)
11 next:
12 cmp ecx, 0 ; проверяем, есть ли еще аргументы
13 jz _end ; если аргументов нет выходим из цикла
14 ; (переход на метку `_end`)
15 pop eax ; иначе извлекаем аргумент из стека
16 call printf ; вызываем функцию печати
17 loop next ; переход к обработке следующего
18 ; аргумента (переход на метку `next`)
19 _end:
20 call quit
```

Рис. 4.7: Рисунок7

Создайте файл lab8-2.asm в каталоге ~/work/arch-pc/lab08 и введите в него текст программы из листинга 8.2. Создайте исполняемый файл и запустите его

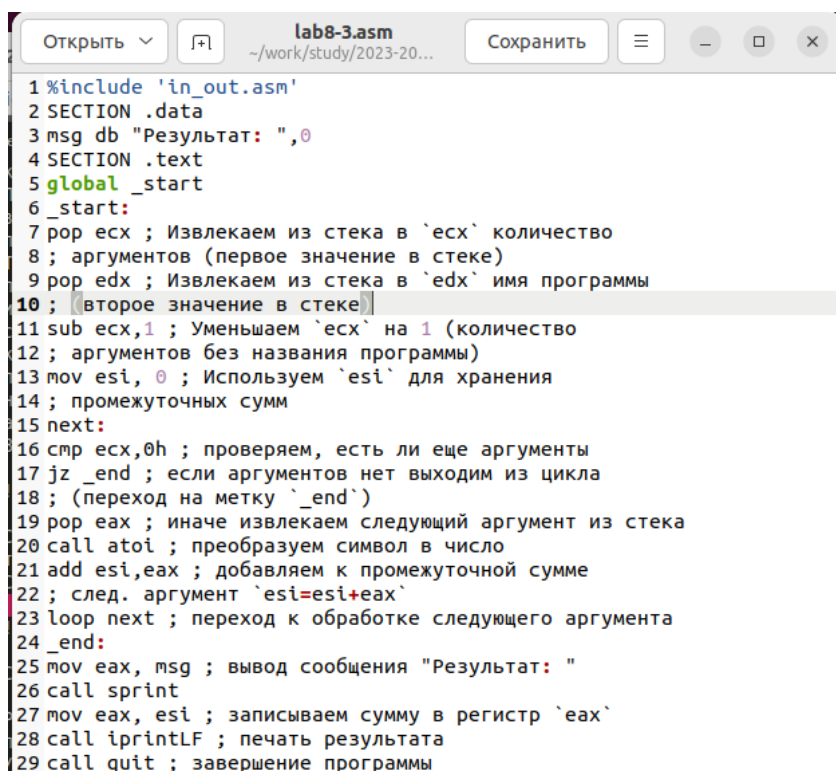
```
alexander@alexander-Swift-SF315-S2G: ~/work/study/2023-2024/Архитектура компьютера
pa/arch-pc/lab08$ touch lab8-2.asm
alexander@alexander-Swift-SF315-S2G: ~/work/study/2023-2024/Архитектура компьютера
pa/arch-pc/lab08$ open lab8-2.asm
alexander@alexander-Swift-SF315-S2G: ~/work/study/2023-2024/Архитектура компьютера
pa/arch-pc/lab08$ nasm -f elf lab8-2.asm
alexander@alexander-Swift-SF315-S2G: ~/work/study/2023-2024/Архитектура компьютера
pa/arch-pc/lab08$ ld -m elf_i386 -o lab8-2 lab8-2.o
alexander@alexander-Swift-SF315-S2G: ~/work/study/2023-2024/Архитектура компьютера
pa/arch-pc/lab08$ ./lab8-2
alexander@alexander-Swift-SF315-S2G: ~/work/study/2023-2024/Архитектура компьютера
pa/arch-pc/lab08$ ./lab8-2 1 2 '3'
1
2
3
```

Рис. 4.8: Рисунок8

Сколько аргументов было обработано программой? -Все

Рассмотрим еще один пример программы которая выводит сумму чисел, которые передаются в программу как аргументы. Создайте файл lab8-3.asm в каталоге ~/work/archpc/lab08 и введите в него текст программы из листинга 8.3.

Создайте исполняемый файл и запустите его, указав аргументы



```
lab8-3.asm
~/work/study/2023-20...
Сохранить

1 %include 'in_out.asm'
2 SECTION .data
3 msg db "Результат: ",0
4 SECTION .text
5 global _start
6 _start:
7 pop ecx ; Извлекаем из стека в `ecx` количество
8 ; аргументов (первое значение в стеке)
9 pop edx ; Извлекаем из стека в `edx` имя программы
10 ; второе значение в стеке
11 sub ecx,1 ; Уменьшаем `ecx` на 1 (количество
12 ; аргументов без названия программы)
13 mov esi, 0 ; Используем `esi` для хранения
14 ; промежуточных сумм
15 next:
16 cmp ecx,0h ; проверяем, есть ли еще аргументы
17 jz _end ; если аргументов нет выходим из цикла
18 ; (переход на метку `_end`)
19 pop eax ; иначе извлекаем следующий аргумент из стека
20 call atoi ; преобразуем символ в число
21 add esi,eax ; добавляем к промежуточной сумме
22 ; след. аргумент `esi=esi+eax`
23 loop next ; переход к обработке следующего аргумента
24 _end:
25 mov eax, msg ; вывод сообщения "Результат: "
26 call sprint
27 mov eax, esi ; записываем сумму в регистр `eax`
28 call iprintLF ; печать результата
29 call quit ; завершение программы
```

Рис. 4.9: Рисунок9

```

alexander@alexander-Swift-SF315-52G:~/work/study/2023-2024/Архитектура компьютера
pa/arch-pc/lab08$ touch lab8-3.asm
alexander@alexander-Swift-SF315-52G:~/work/study/2023-2024/Архитектура компьютера
pa/arch-pc/lab08$ open lab8-3.asm
alexander@alexander-Swift-SF315-52G:~/work/study/2023-2024/Архитектура компьютера
pa/arch-pc/lab08$ nasm -f elf lab8-3.asm
alexander@alexander-Swift-SF315-52G:~/work/study/2023-2024/Архитектура компьютера
pa/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
alexander@alexander-Swift-SF315-52G:~/work/study/2023-2024/Архитектура компьютера
pa/arch-pc/lab08$ ./lab8-3 12 13 7 10 5
Результат: 47
alexander@alexander-Swift-SF315-52G:~/work/study/2023-2024/Архитектура компьютера

```

Рис. 4.10: Рисунок10

Измените текст программы из листинга 8.3 для вычисления произведения аргументов командной строки.

```

1 %include 'in_out.asm'
2 SECTION .data
3 msg db "Результат: ",0
4 SECTION .text
5 global _start
6 _start:
7 pop ecx ; Извлекаем из стека в `ecx` количество
8 ; аргументов (первое значение в стеке)
9 pop edx ; Извлекаем из стека в `edx` имя программы
10 ; (второе значение в стеке)
11 sub ecx,1 ; Уменьшаем `ecx` на 1 (количество
12 ; аргументов без названия программы)
13 mov esi, 1 ; Используем `esi` для хранения
14 ; произведения
15 next:
16 cmp ecx,0h ; проверяем, есть ли еще аргументы
17 jz _end ; если аргументов нет выходим из цикла
18 ; (переход на метку `_end`)
19 pop eax ; иначе извлекаем следующий аргумент из стека
20 call atoi ; преобразуем символ в число
21 mul esi ; добавляем к промежуточной произведению
22 mov esi, eax
23 ; след. аргумент `esi=esi*eax`
24 loop next ; переход к обработке следующего аргумента
25 _end:
26 mov eax, msg ; вывод сообщения "Результат: "
27 call sprint
28 mov eax, esi ; записываем сумму в регистр `eax`
29 call iprintf ; печать результата
30 call quit ; завершение программы

```

Рис. 4.11: Рисунок11

```

pa/arch-pc/lab08$ nasm -f elf lab8-3.asm
alexander@alexander-Swift-SF315-52G:~/work/study/2023-2024/Архитектура компьютера
pa/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
alexander@alexander-Swift-SF315-52G:~/work/study/2023-2024/Архитектура компьютера
pa/arch-pc/lab08$ ./lab8-3 12 13 7 10 5
Результат: 54600
alexander@alexander-Swift-SF315-52G:~/work/study/2023-2024/Архитектура компьютера
pa/arch-pc/lab08$

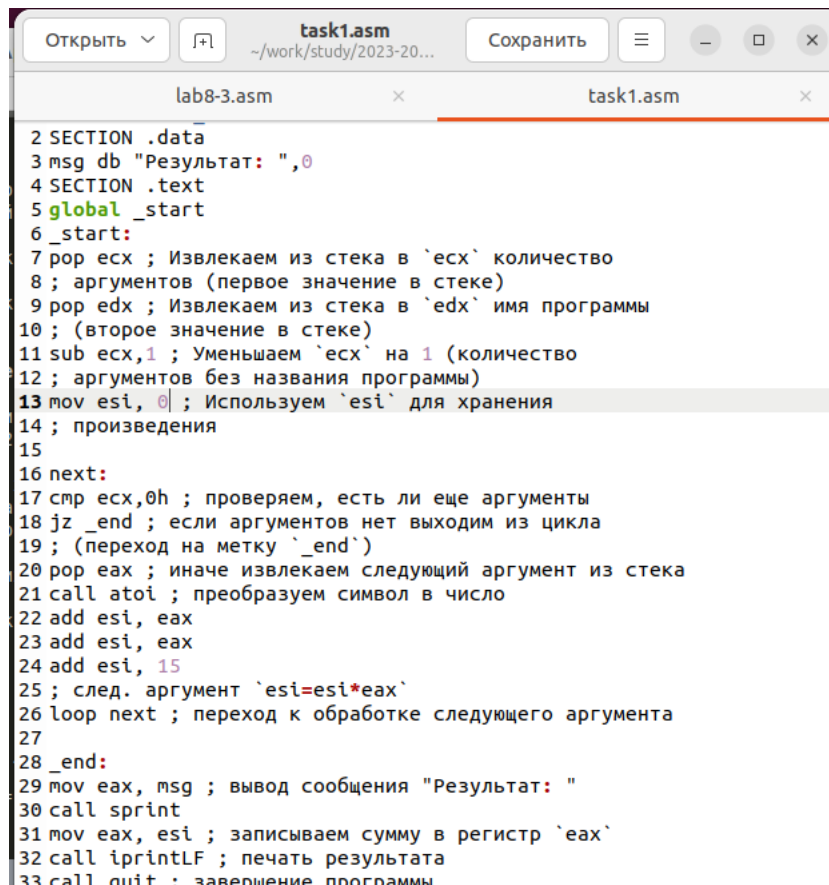
```

Рис. 4.12: Рисунок12

## 5 Задание для самостоятельной работы

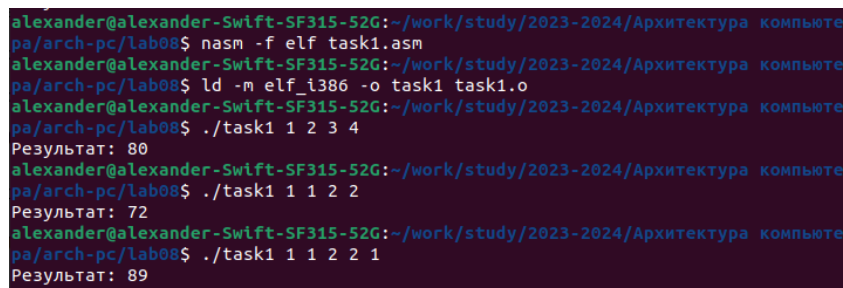
1. Напишите программу, которая находит сумму значений функции  $f(x)$  для  $x = x_1, x_2, \dots, x_n$ , т.е. программа должна выводить значение  $f(x_1) + f(x_2) + \dots + f(x_n)$ . Значения  $x_i$  передаются как аргументы. Вид функции  $f(x)$  выбрать из таблицы 8.1 вариантов заданий в соответствии с вариантом, полученным при выполнении лабораторной работы № 7. Создайте исполняемый файл и проверьте его работу на нескольких наборах  $x = x_1, x_2, \dots, x_n$ .





```
2 SECTION .data
3 msg db "Результат: ",0
4 SECTION .text
5 global _start
6 _start:
7 pop ecx ; Извлекаем из стека в `ecx` количество
8 ; аргументов (первое значение в стеке)
9 pop edx ; Извлекаем из стека в `edx` имя программы
10 ; (второе значение в стеке)
11 sub ecx,1 ; Уменьшаем `ecx` на 1 (количество
12 ; аргументов без названия программы)
13 mov esi, 0 ; Используем `esi` для хранения
14 ; произведения
15
16 next:
17 cmp ecx,0h ; проверяем, есть ли еще аргументы
18 jz _end ; если аргументов нет выходим из цикла
19 ; (переход на метку `_end`)
20 pop eax ; иначе извлекаем следующий аргумент из стека
21 call atoi ; преобразуем символ в число
22 add esi, eax
23 add esi, eax
24 add esi, 15
25 ; след. аргумент `esi=esi*eax`
26 loop next ; переход к обработке следующего аргумента
27
28 _end:
29 mov eax, msg ; вывод сообщения "Результат: "
30 call printf
31 mov eax, esi ; записываем сумму в регистр `eax`
32 call printf ; печать результата
33 call _exit ; завершение программы
```

Рис. 5.1: Рисунок13



```
alexander@alexander-Swift-SF315-52G:~/work/study/2023-2024/Архитектура компьюте
pa/arch-pc/lab08$ nasm -f elf task1.asm
alexander@alexander-Swift-SF315-52G:~/work/study/2023-2024/Архитектура компьюте
pa/arch-pc/lab08$ ld -m elf_i386 -o task1 task1.o
alexander@alexander-Swift-SF315-52G:~/work/study/2023-2024/Архитектура компьюте
pa/arch-pc/lab08$ ./task1 1 2 3 4
Результат: 80
alexander@alexander-Swift-SF315-52G:~/work/study/2023-2024/Архитектура компьюте
pa/arch-pc/lab08$ ./task1 1 1 2 2
Результат: 72
alexander@alexander-Swift-SF315-52G:~/work/study/2023-2024/Архитектура компьюте
pa/arch-pc/lab08$ ./task1 1 1 2 2 1
Результат: 89
```

Рис. 5.2: Рисунок14

Опять же вариант 1.

## 6 Выводы

Мы научились писать программы с циклами, также обрабатывать аргументы командной строки и работать со стеком.

## **Список литературы**