

Отчёт по лабораторной работе 8

Дисциплина: архитектура компьютера

Айдарбекова Алия НММбд-01-23

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	8
4.1	Реализация циклов в NASM	8
4.2	Обработка аргументов командной строки	14
4.3	Выполнение заданий для самостоятельной работы	19
5	Выводы	22

Список иллюстраций

4.1	Изменение кода lab8-1.asm	9
4.2	Компиляция текста программы lab8-1.asm	10
4.3	Изменение кода lab8-1.asm	11
4.4	Компиляция текста программы lab8-1.asm	12
4.5	Изменение кода lab8-1.asm	13
4.6	Компиляция текста программы lab8-1.asm	14
4.7	Изменение кода lab8-2.asm	15
4.8	Компиляция текста программы lab8-2.asm	16
4.9	Изменение кода lab8-3.asm	17
4.10	Компиляция текста программы lab8-3.asm	17
4.11	Изменение кода lab8-3.asm	18
4.12	Компиляция текста программы lab8-3.asm	19
4.13	Изменение кода prog.asm	20
4.14	Компиляция текста программы prog.asm	21

Список таблиц

1 Цель работы

Целью работы является приобретение навыков написания программ с использованием циклов и обработкой аргументов командной строки..

2 Задание

1. Изучение циклов в ассемблере
2. Изучение стека в ассемблере
3. Изучение передачи аргументов в ассемблере
4. Выполнение заданий, рассмотрение примеров
5. Выполнение заданий для самостоятельной работы

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.

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.

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.

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.

4 Выполнение лабораторной работы

4.1 Реализация циклов в NASM

Создаю каталог для выполнения лабораторной работы № 8 и файл с именем lab8-1.asm.

При использовании инструкции `loop` в NASM для реализации циклов, нужно помнить о том, что данная инструкция использует регистр `ecx` в качестве счетчика и на каждой итерации уменьшает его значение на единицу.

Рассмотрим пример программы, которая выводит значение регистра `ecx`. Написала текст программы из листинга 8.1 в файле lab8-1.asm. (рис. [4.1])



```
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
```


Рис. 4.1: Изменение кода lab8-1.asm

Затем создаю исполняемый файл и проверю его работу.(рис. [4.2])

```
[arayjdarbekova@fedora lab08]$  
[arayjdarbekova@fedora lab08]$ nasm -f elf lab8-1.asm  
[arayjdarbekova@fedora lab08]$ ld -m elf_i386 lab8-1.o -o lab8-1  
[arayjdarbekova@fedora lab08]$ ./lab8-1  
Введите N: 3  
3  
2  
1  
[arayjdarbekova@fedora lab08]$
```

Рис. 4.2: Компиляция текста программы lab8-1.asm

В данном примере демонстрируется, что использование регистра есх в инструкции loop может привести к неправильной работе программы. В тексте программы вношу изменения, которые включают изменение значения регистра есх внутри цикла.(рис. [4.3])

Открыть ▾ 

lab8-1.asm
~/work/lab08

```
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|
```

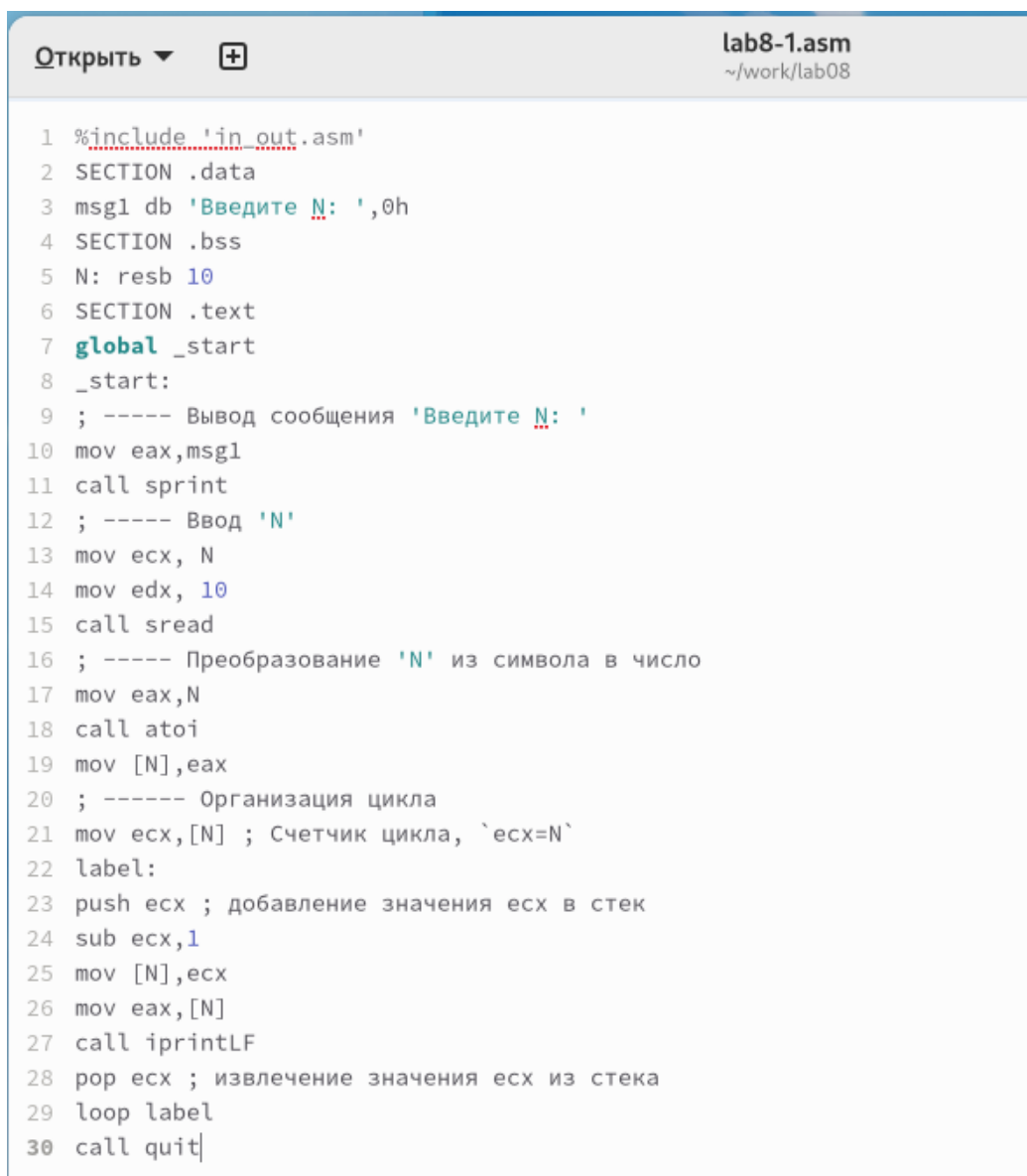
Рис. 4.3: Изменение кода lab8-1.asm

Программа запускает бесконечный цикл при нечетном значении N и выводит только нечетные числа при четном значении N. (рис. [4.4])

```
4294895212
4294895210
4294895208
4294895206
4294895204
4294895202
4294895200
4294895198
4294895^C
[arayjdarbekova@fedora lab08]$ ./lab8-1
Введите N: 4
3
1
[arayjdarbekova@fedora lab08]$
```

Рис. 4.4: Компиляция текста программы lab8-1.asm

Для того чтобы использовать регистр `ecx` в цикле и обеспечить корректность работы программы, применяется стек. Вношу изменения в текст программы, добавив команды `push` и `pop` для сохранения значения счётчика цикла `loop` в стеке.(рис. [4.5])



```
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 ; добавление значения ecx в стек
24 sub ecx,1
25 mov [N],ecx
26 mov eax,[N]
27 call iprintLF
28 pop ecx ; извлечение значения ecx из стека
29 loop label
30 call quit
```

Рис. 4.5: Изменение кода lab8-1.asm


Был создан исполняемый файл и проверена его работа. Программа выводит числа от N-1 до 0, где количество проходов цикла соответствует значению N.(рис. [4.6])

```
[arayjdarbekova@fedora lab08]$  
[arayjdarbekova@fedora lab08]$ nasm -f elf lab8-1.asm  
[arayjdarbekova@fedora lab08]$ ld -m elf_i386 lab8-1.o -o lab8-1  
[arayjdarbekova@fedora lab08]$ ./lab8-1  
Введите N: 3  
2  
1  
0  
[arayjdarbekova@fedora lab08]$ ./lab8-1  
Введите N: 4  
3  
2  
1  
0  
[arayjdarbekova@fedora lab08]$
```

Рис. 4.6: Компиляция текста программы lab8-1.asm

4.2 Обработка аргументов командной строки

Я создала файл lab8-2.asm в каталоге ~/work/arch-рс/lab08 и ввела в него текст программы из листинга 8.2. (рис. [4.7])

Открыть ▾ 

lab8-2.asm
~/work/lab08

```
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 sprintf ; вызываем функцию печати
17 loop next ; переход к обработке следующего|
18 ; аргумента (переход на метку `next`)
19 _end:
20 call quit
```

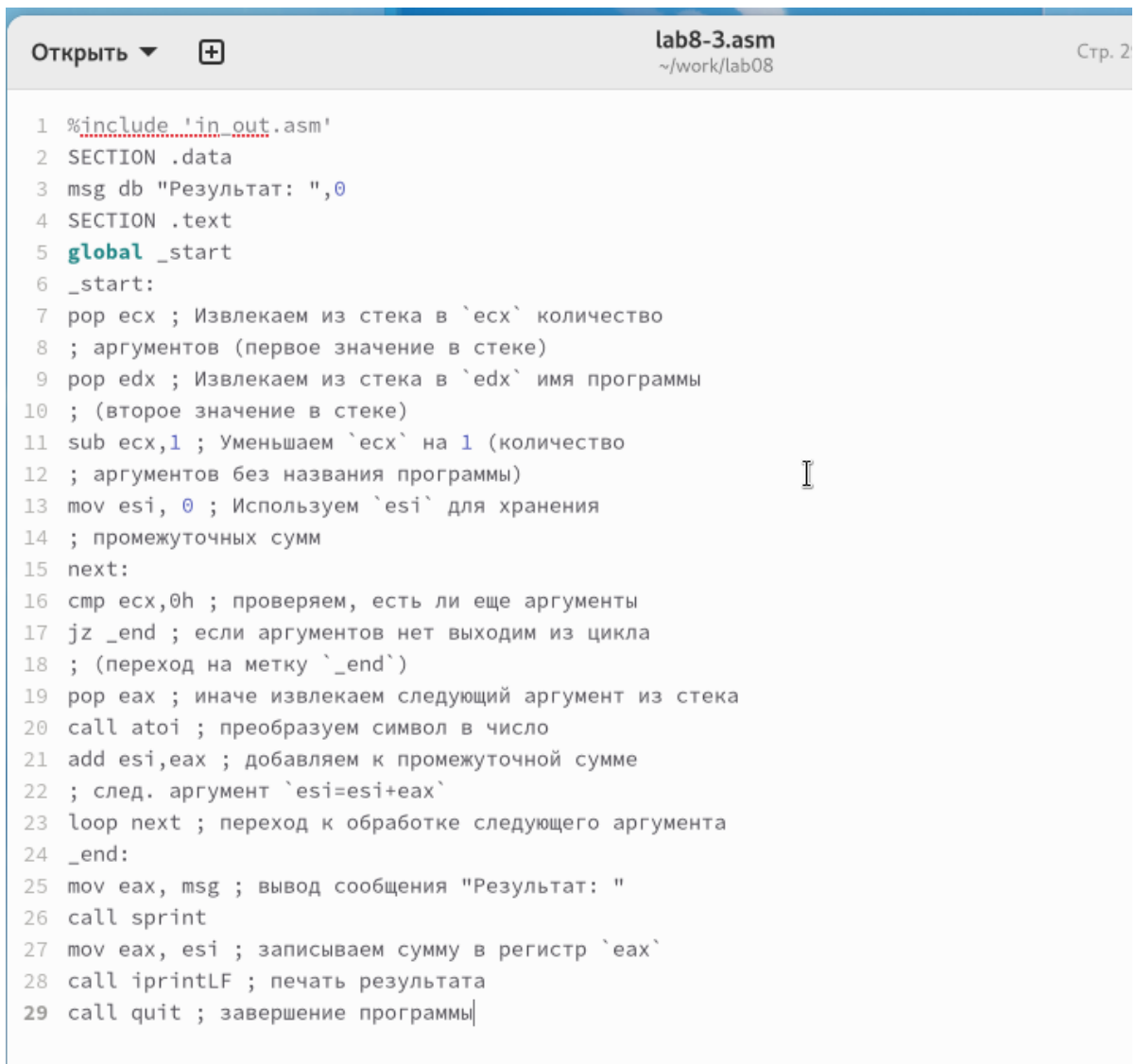
Рис. 4.7: Изменение кода lab8-2.asm

Затем я создала исполняемый файл и запустила его, указав аргументы. Программа успешно обработала 5 аргументов. Аргументами считаются слова/числа, разделенные пробелом. (рис. [4.8])

```
[arayjdarbekova@fedora lab08]$  
[arayjdarbekova@fedora lab08]$ nasm -f elf lab8-2.asm  
[arayjdarbekova@fedora lab08]$ ld -m elf_i386 lab8-2.o -o lab8-2  
[arayjdarbekova@fedora lab08]$ ./lab8-2  
[arayjdarbekova@fedora lab08]$ ./lab8-2 argument 1 argument 2 'argument 3'  
argument  
1  
argument  
2  
argument 3  
[arayjdarbekova@fedora lab08]$
```

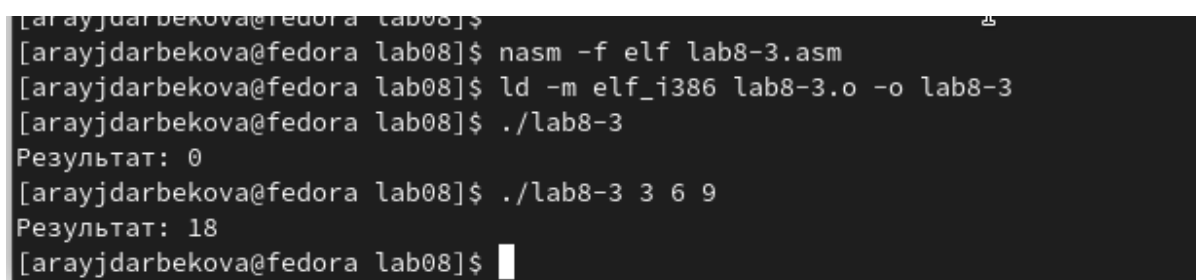
Рис. 4.8: Компиляция текста программы lab8-2.asm

Теперь рассмотрим ещё один пример программы, которая выводит сумму чисел, переданных в программу как аргументы командной строки. (рис. [4.9]) (рис. [4.10])



```
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
```

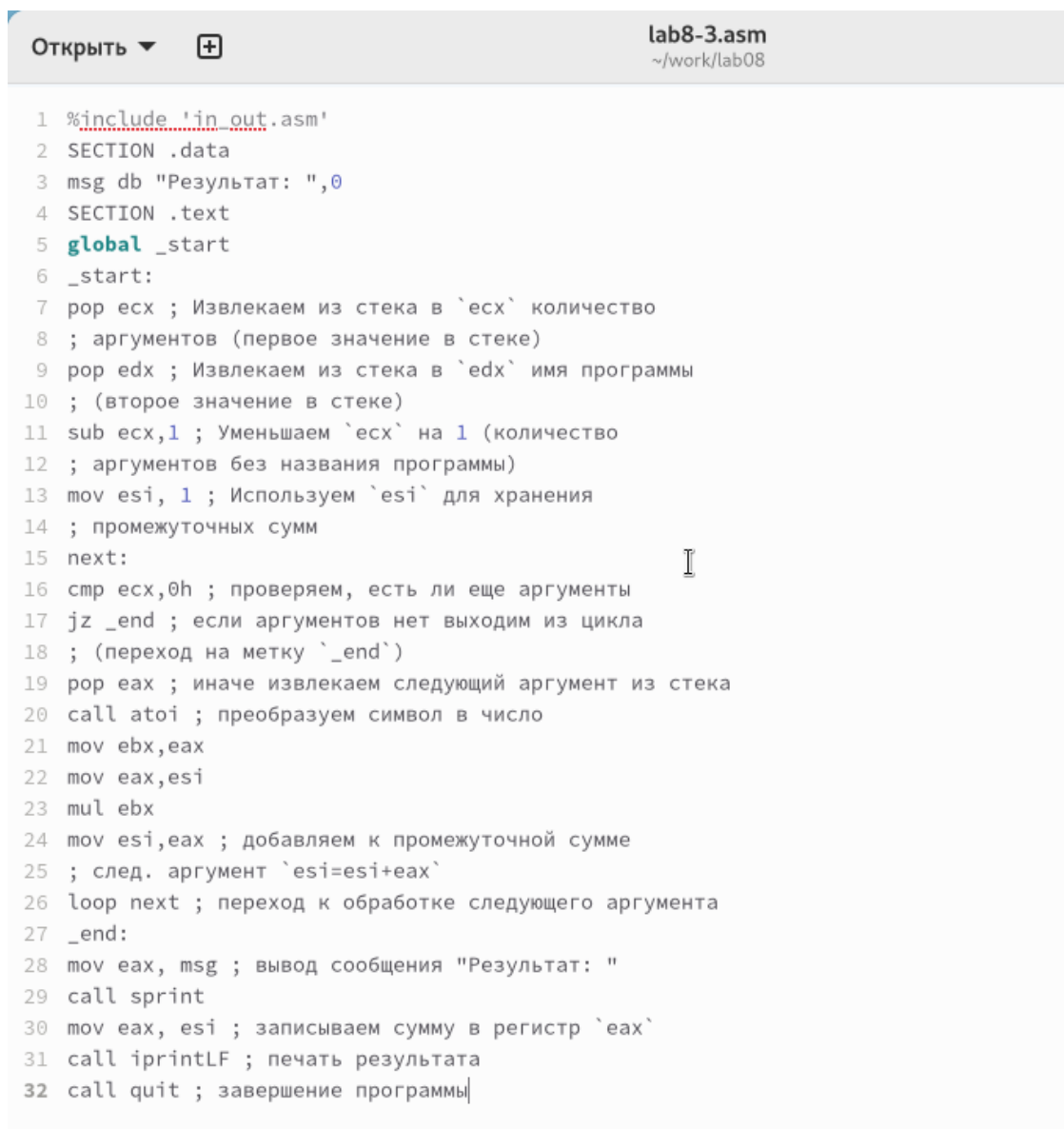
Рис. 4.9: Изменение кода lab8-3.asm



```
[arayjdarbekova@fedora lab08]$
[arayjdarbekova@fedora lab08]$ nasm -f elf lab8-3.asm
[arayjdarbekova@fedora lab08]$ ld -m elf_i386 lab8-3.o -o lab8-3
[arayjdarbekova@fedora lab08]$ ./lab8-3
Результат: 0
[arayjdarbekova@fedora lab08]$ ./lab8-3 3 6 9
Результат: 18
[arayjdarbekova@fedora lab08]$
```

Рис. 4.10: Компиляция текста программы lab8-3.asm

Я изменила текст программы из листинга 8.3 для вычисления произведения аргументов командной строки. (рис. [4.11]) (рис. [4.12])



```
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 ; 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 mov ebx,eax
22 mov eax,esi
23 mul ebx
24 mov esi,eax ; добавляем к промежуточной сумме
25 ; след. аргумент `esi=esi+eax`
26 loop next ; переход к обработке следующего аргумента
27 _end:
28 mov eax,msg ; вывод сообщения "Результат: "
29 call sprint
30 mov eax,esi ; записываем сумму в регистр `eax`
31 call iprintLF ; печать результата
32 call quit ; завершение программы
```

Рис. 4.11: Изменение кода lab8-3.asm

```
[arayjdarbekova@fedora lab08]$  
[arayjdarbekova@fedora lab08]$ nasm -f elf lab8-3.asm  
[arayjdarbekova@fedora lab08]$ ld -m elf_i386 lab8-3.o -o lab8-3  
[arayjdarbekova@fedora lab08]$ ./lab8-3  
Результат: 1  
[arayjdarbekova@fedora lab08]$ ./lab8-3 3 6 9  
Результат: 162  
[arayjdarbekova@fedora lab08]$
```


Рис. 4.12: Компиляция текста программы lab8-3.asm

4.3 Выполнение заданий для самостоятельной работы

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 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 . (рис. [4.13]) (рис. [4.14])

Мой вариант 14:

$$f(x) = 7(x + 1)$$

Открыть ▾ 

prog.asm
~/work/lab08

```
1 %include 'in_out.asm'
2 SECTION .data
3 msg db "Результат: ",0
4 fx: db 'f(x)= 7(x + 1)',0
5
6 SECTION .text
7 global _start
8 _start:
9 mov eax, fx
10 call sprintLF
11 pop ecx
12 pop edx
13 sub ecx,1
14 mov esi, 0
15
16 next:
17 cmp ecx,0h
18 jz _end
19 pop eax
20 call atoi
21 add eax,1
22 mov ebx,7
23 mul ebx
24 add esi,eax
25
26 loop next
27
28 _end:
29 mov eax, msg
30 call sprint
31 mov eax, esi
32 call iprintLF
33 call quit
```

Рис. 4.13: Изменение кода prog.asm

```
[arayjdarbekova@fedora lab08]$ nasm -f elf prog.asm
[arayjdarbekova@fedora lab08]$ ld -m elf_i386 prog.o -o prog
[arayjdarbekova@fedora lab08]$ ./prog
f(x)= 7(x + 1)
Результат: 0
[arayjdarbekova@fedora lab08]$ ./prog 1
f(x)= 7(x + 1)
Результат: 14
[arayjdarbekova@fedora lab08]$ ./prog 3 6 9
f(x)= 7(x + 1)
Результат: 147
[arayjdarbekova@fedora lab08]$
```

Рис. 4.14: Компиляция текста программы prog.asm

5 Выводы

Освоили работы со стеком, циклом и аргументами на ассемблере `naasm`.