

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: Ибрагимов Гаджимурад Шамильевич

Группа: НКАбд-02-25

МОСКВА

2025 г.

СОДЕРЖАНИЕ

Цель работы - - - - -	3
Теоретическое введение - - - - -	4
Выполнение лабораторной работы - - - - -	5
Выполнение самостоятельной работы - - - - -	12
Вывод - - - - -	14
Источники - - - - -	15

1. ЦЕЛЬ РАБОТЫ

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

2. ТЕОРЕТИЧЕСКОЕ ВВЕДЕНИЕ

2.1 Сущность стека

Стек (англ. stack — "стопка") — это структура данных, работающая по принципу LIFO (Last In, First Out) — "последним пришёл, первым ушёл". Реализация стека приведена во многих языках программирования.

2.2 Функционал стека

С данной структурой данных возможен ограниченный список действий:

Основные операции со стеком:

1. `push(x)` — добавить элемент `x` на вершину стека.
2. `pop()` — удалить верхний элемент.
3. `top()` — возвращает верхний элемент без удаления.
4. `empty()` — проверка на пустоту (возвращает `true/false`)

Реализация стека напоминает чипсы Pringles;)

2.3 Циклы

Для организации циклов существуют специальные инструкции. Для всех инструкций максимальное количество проходов задаётся в регистре `есх`. Наиболее простой является инструкция `Loop`. Она позволяет организовать безусловный цикл, типичная структура которого имеет следующий вид:

```
    mov    ecx, 100    ; Количество проходов
NextStep:
    ...
    ...              ; тело цикла
    ...
loop  NextStep    ; Повторить 'есх' раз от метки NextStep
```

Инструкция `loop` выполняется в два этапа. Сначала из регистра `есх` вычитается единица и его значение сравнивается с нулем. 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`.

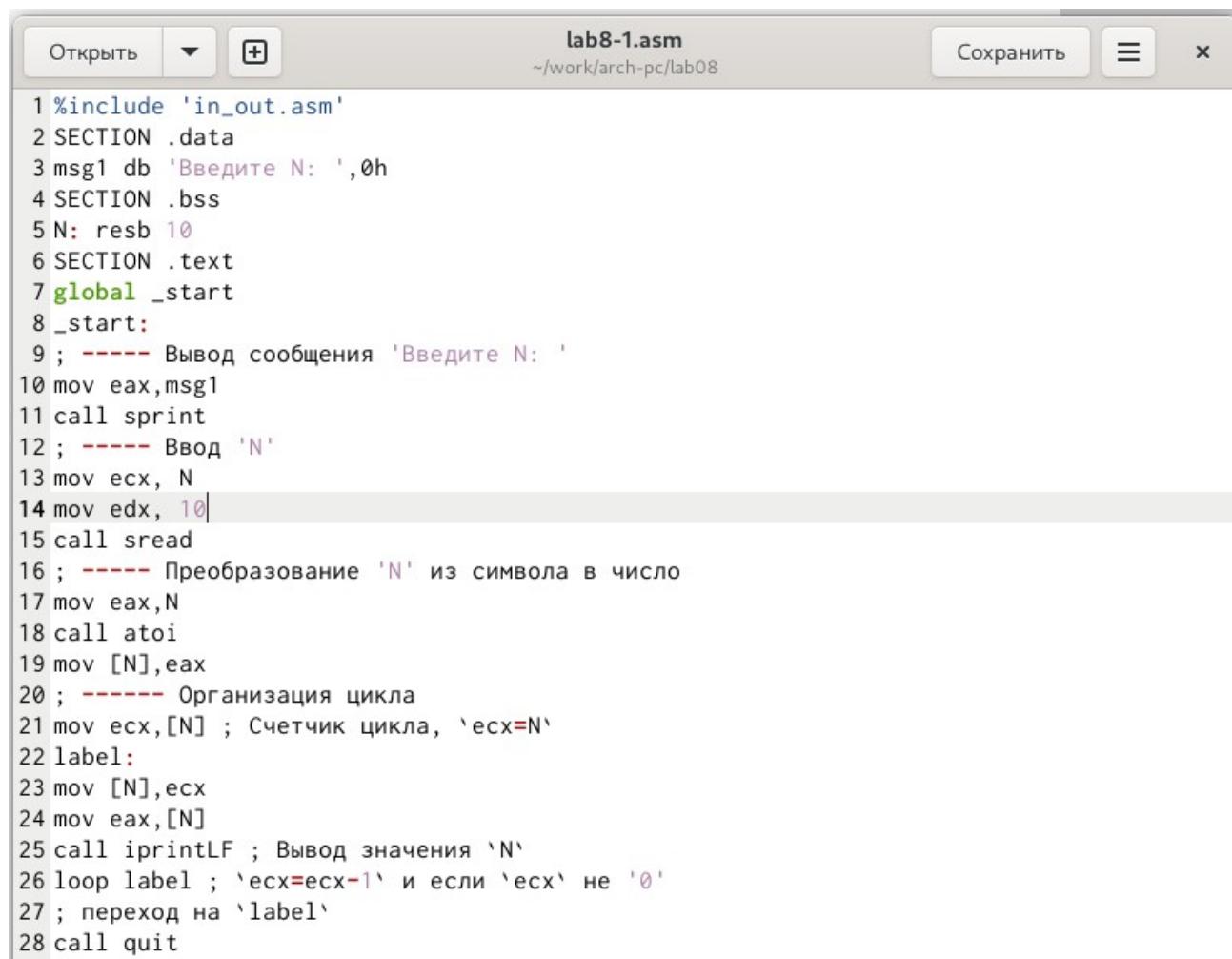
3. ВЫПОЛНЕНИЕ ЛАБОРАТОРНОЙ РАБОТЫ

3.1.1 Реализация циклов в NASM

Создаем каталог для программ лабораторной работы № 8, переходим в него и создаём файл lab8-1.asm

```
gsibragimov@dk1n22 ~/work/arch-pc/lab08 $ touch lab8-1.asm
gsibragimov@dk1n22 ~/work/arch-pc/lab08 $
```

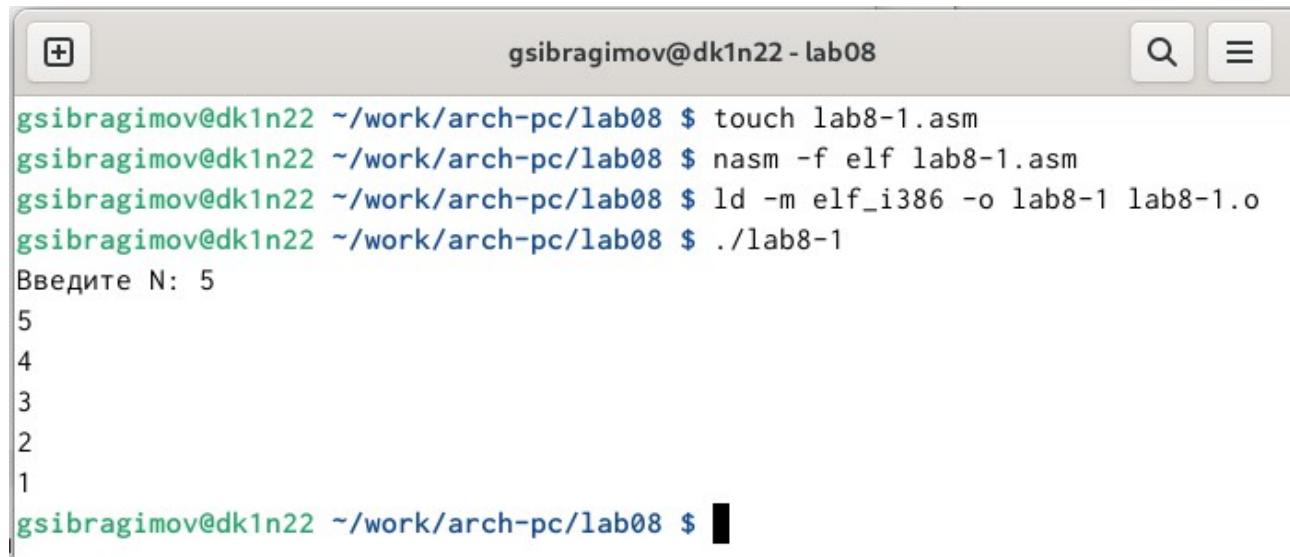
Реализация цикла с N числом итераций с последующим уменьшением числа на единицу:



The screenshot shows a text editor window titled "lab8-1.asm" with the path "~/work/arch-pc/lab08". The file contains the following assembly code:

```
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
```

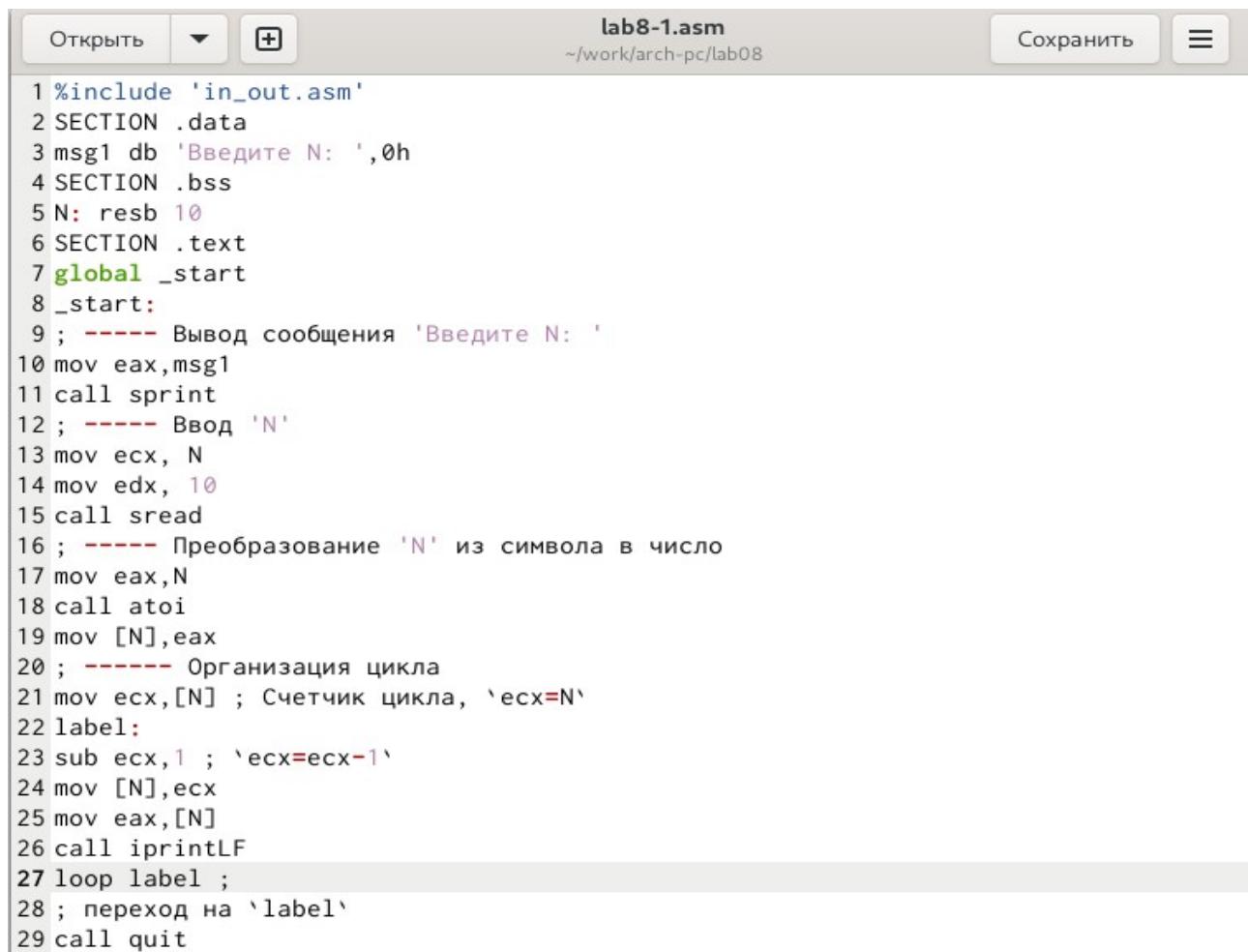
Запустим программу и посмотрим на результат:



```
gsibragimov@dk1n22 ~/work/arch-pc/lab08 $ touch lab8-1.asm
gsibragimov@dk1n22 ~/work/arch-pc/lab08 $ nasm -f elf lab8-1.asm
gsibragimov@dk1n22 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-1 lab8-1.o
gsibragimov@dk1n22 ~/work/arch-pc/lab08 $ ./lab8-1
Введите N: 5
5
4
3
2
1
gsibragimov@dk1n22 ~/work/arch-pc/lab08 $
```

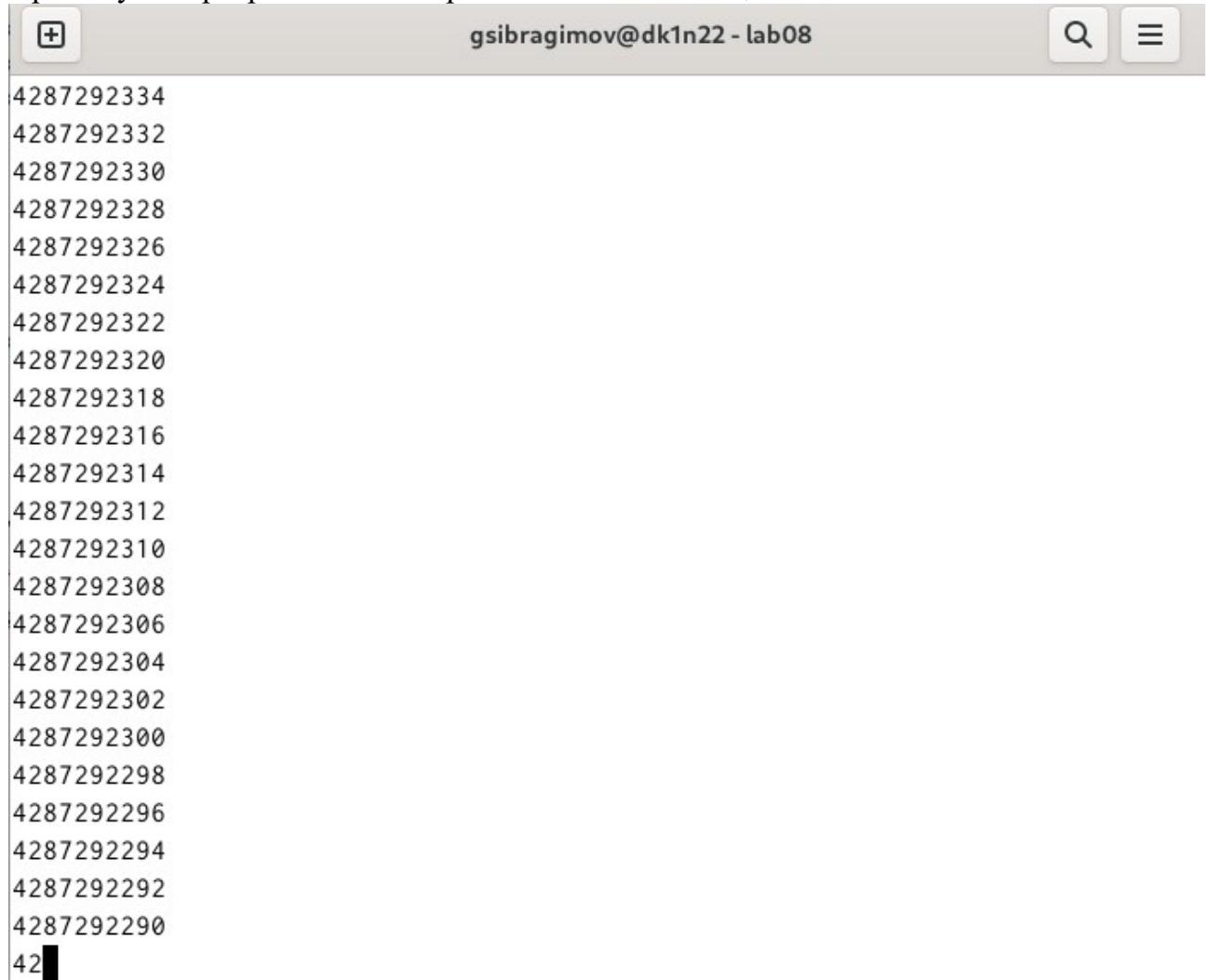
3.1.2 Интересный пример

Данный пример показывает, что использование регистра ecx в теле цикла loop может привести к некорректной работе программы. Изменим текст программы добавив и изменив значение регистра ecx в цикле:



```
Открыть ▾ + lab8-1.asm
~/work/arch-pc/lab08 Сохранить ⌂
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
```

При запуске программы нас встречает бесконечный цикл:



A screenshot of a terminal window titled "gsibragimov@dk1n22 - lab08". The window contains a vertical list of approximately 30 identical numbers, each consisting of six digits. The numbers are: 4287292334, 4287292332, 4287292330, 4287292328, 4287292326, 4287292324, 4287292322, 4287292320, 4287292318, 4287292316, 4287292314, 4287292312, 4287292310, 4287292308, 4287292306, 4287292304, 4287292302, 4287292300, 4287292298, 4287292296, 4287292294, 4287292292, 4287292290, and 42. A cursor is visible at the end of the last number, indicating the program is still running.

```
4287292334
4287292332
4287292330
4287292328
4287292326
4287292324
4287292322
4287292320
4287292318
4287292316
4287292314
4287292312
4287292310
4287292308
4287292306
4287292304
4287292302
4287292300
4287292298
4287292296
4287292294
4287292292
4287292290
42
```

3.1.3 Для восстановления работоспособности программы изменим код:

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

The screenshot shows a text editor window with the following details:

- File menu: Открыть (Open), Сохранить (Save).
- File path: lab8-1.asm, ~ /work/arch-pc/lab08.
- Code content:

```
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 ; переход на 'label'
31 call quit
```

Запустим программу:

The terminal window shows the following session:

```
gsibragimov@dk1n22 ~ /work/arch-pc/lab08 $ nasm -f elf lab8-1.asm
gsibragimov@dk1n22 ~ /work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-1 lab8-1.o
gsibragimov@dk1n22 ~ /work/arch-pc/lab08 $ ./lab8-1
Введите N: 5
4
3
2
1
0
gsibragimov@dk1n22 ~ /work/arch-pc/lab08 $ ./lab8-1
Введите N: 3
2
1
0
```

Число проходов соответствует счетчику цикла N введенному с клавиатуры.

3.2.1 Обработка аргументов командной строки

Создаем файл lab8-2.asm в нашем рабочем каталоге и вводим в него следующий текст программы:

```
gsibragimov@dk1n22 ~/work/arch-pc/lab08 $ ls
in_out.asm  lab8-1  lab8-1.asm  lab8-1.o  lab8-2  lab8-2.asm  lab8-2.o
gsibragimov@dk1n22 ~/work/arch-pc/lab08 $
```

Создаем исполняемый файл и запускаем его, указав аргументы следующие:

аргумент1 аргумент2 'аргумент3'

```
gsibragimov@dk1n22 ~/work/arch-pc/lab08 $ ./lab8-2 аргумент1 аргумент2 'аргумент3'
аргумент1
аргумент
2
аргумент 3
gsibragimov@dk1n22 ~/work/arch-pc/lab08 $ █
```

3.3.1 Программа вычисления суммы аргументов командной строки

Рассмотрим еще один пример программы, которая выводит сумму чисел, которые передаются в программу как аргументы.

Создаем файл lab8-3.asm

```
gsibragimov@dk3n55 ~/work/arch-pc/lab08 $ ls
in_out.asm  lab8-1.asm  lab8-2      lab8-2.o
lab8-1      lab8-1.o   lab8-2.asm  lab8-3.asm
gsibragimov@dk3n55 ~/work/arch-pc/lab08 $ █
```

The screenshot shows a text editor window with the following details:

- File menu: Открыть (Open), Сохранить (Save).
- Title bar: lab8-3.asm, ~/work/arch-pc/lab08.
- Code area:

```
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
```

Создаем исполняемый файл и запускаем его:

```
gsibragimov@dk3n55 ~/work/arch-pc/lab08 $ nasm -f elf lab8-3.asm
gsibragimov@dk3n55 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-3 lab8-3.o
gsibragimov@dk3n55 ~/work/arch-pc/lab08 $ ./lab8-3
Результат: 0
gsibragimov@dk3n55 ~/work/arch-pc/lab08 $ ./lab8-3 12 13 7 10 5
Результат: 47
gsibragimov@dk3n55 ~/work/arch-pc/lab08 $ 
```

3.3.2 Давайте изменим текст программы для вычисления произведения аргументов командной строки.

The screenshot shows a window titled "lab8-3.asm — KWrite". The menu bar includes "Файл", "Правка", "Выделение", "Вид", "Переход", "Сервис", "Настройка", and "Справка". Below the menu is a toolbar with icons for "Создать", "Открыть...", "Сохранить", and "Сохранить как...". The code editor displays the following assembly code:

```
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
; аргументов (первое значение в стеке)
9 pop edx ; Извлекаем из стека в 'edx' имя программы
; (второе значение в стеке)
11 sub ecx,1 ; Уменьшаем 'ecx' на 1 (количество
; аргументов без названия программы)
13 mov esi,1 ; Используем 'esi' для хранения
; промежуточных значений
15 next:
16 cmp ecx,0h ; проверяем, есть ли еще аргументы
17 jz _end ; если аргументов нет выходим из цикла
; (переход на метку '_end')
19 pop eax ; иначе извлекаем следующий аргумент из стека
20 call atoi ; преобразуем символ в число
21 imul esi, eax
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
```

Запускаем программу, указав аргументы:

```
gsibragimov@dk3n55 ~/work/arch-pc/lab08 $ nasm -f elf lab8-3.asm
gsibragimov@dk3n55 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-3 lab8-3.o
gsibragimov@dk3n55 ~/work/arch-pc/lab08 $ ./lab8-3 2 1 3
Результат: 6
gsibragimov@dk3n55 ~/work/arch-pc/lab08 $
```

Результат есть произведение аргументов

4. ВЫПОЛНЕНИЕ ЛАБОРАТОРНОЙ РАБОТЫ

4.1. Напишите программу, которая находит сумму значений функции $f(x)$ для $x = x_1, x_2, \dots, x_n$, т.е. программа должна выводить значение $f(x_1) + f(x_2) + \dots + f(x_n)$.

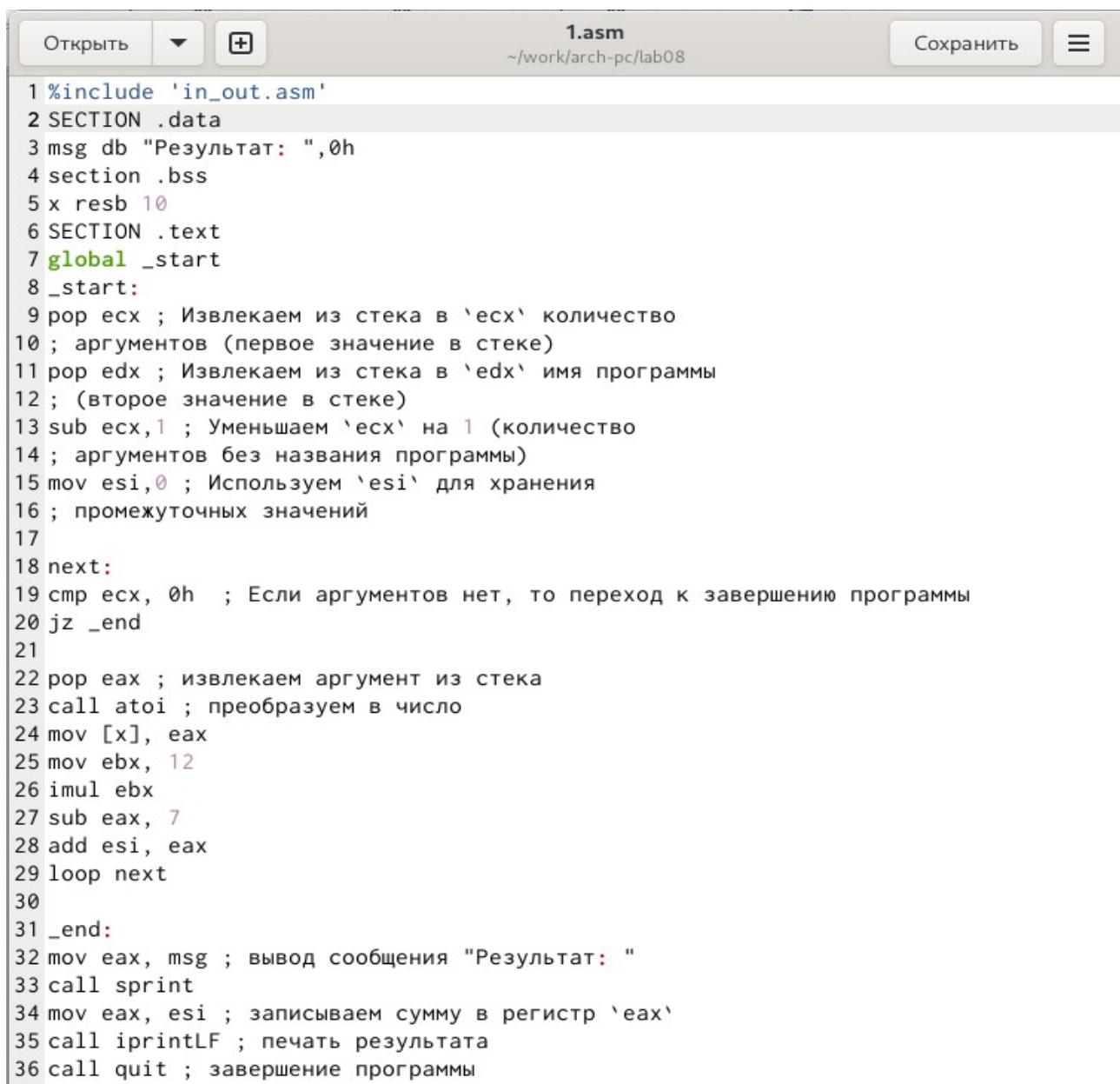
Значения x_i передаются как аргументы.

Вид функции $f(x)$: **13** $12x - 7$

Создаем файл 1.asm

```
gsibragimov@dk3n55 ~/work/arch-pc/lab08 $ touch 1.asm
```

Текст программы:



The screenshot shows a text editor window with the following details:

- File menu: Открыть (Open), Сохранить (Save).
- Title bar: 1.asm, ~/work/arch-pc/lab08.
- Code area:

```
1 %include 'in_out.asm'
2 SECTION .data
3 msg db "Результат: ",0h
4 section .bss
5 x resb 10
6 SECTION .text
7 global _start
8 _start:
9 pop ecx ; Извлекаем из стека в 'ecx' количество
10 ; аргументов (первое значение в стеке)
11 pop edx ; Извлекаем из стека в 'edx' имя программы
12 ; (второе значение в стеке)
13 sub ecx,1 ; Уменьшаем 'ecx' на 1 (количество
14 ; аргументов без названия программы)
15 mov esi,0 ; Используем 'esi' для хранения
16 ; промежуточных значений
17
18 next:
19 cmp ecx, 0h ; Если аргументов нет, то переход к завершению программы
20 jz _end
21
22 pop eax ; извлекаем аргумент из стека
23 call atoi ; преобразуем в число
24 mov [x], eax
25 mov ebx, 12
26 imul ebx
27 sub eax, 7
28 add esi, eax
29 loop next
30
31 _end:
32 mov eax, msg ; вывод сообщения "Результат: "
33 call sprint
34 mov eax, esi ; записываем сумму в регистр 'eax'
35 call iprintLF ; печать результата
36 call quit ; завершение программы
```

Создаем исполняемый файл и проверяем его работу на нескольких значениях:

```
gsibragimov@dk3n55 ~/work/arch-pc/lab08 $ nasm -f elf 1.asm
gsibragimov@dk3n55 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o 1 1.o
gsibragimov@dk3n55 ~/work/arch-pc/lab08 $ ./1
Результат: 0
gsibragimov@dk3n55 ~/work/arch-pc/lab08 $ ./1 1 2 3 4 5
Результат: 145
gsibragimov@dk3n55 ~/work/arch-pc/lab08 $ ./1 2 2 2
Результат: 51
gsibragimov@dk3n55 ~/work/arch-pc/lab08 $ ./1 3 3
Результат: 58
gsibragimov@dk3n55 ~/work/arch-pc/lab08 $
```

5. ВЫВОД

В результате выполнения данной лабораторной работы мы научились пользоваться циклами для решения задач, а также познакомились со структурой данных “стек”.

6. ИСТОЧНИКИ

1. 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.
2. Таненбаум Э. Архитектура компьютера. — 6-е изд. — СПб. : Питер, 2013. — 874 с.