

Лабораторная работа №8

Программирование цикла. Обработка аргументов командной строки

Четвергова Мария Викторовна

Содержание

1	Цель работы	5
2	Теоретическое введение	6
3	Выполнение лабораторной работы	8
4	Выводы	20

Список иллюстраций

3.1	Рис.1 Создание каталога и файла для работы	8
3.2	Ввод листинга программы	9
3.3	Проверка работы данной программы	10
3.4	Изменение листинга программы	11
3.5	Проверка работы изменённой программы	11
3.6	Изменение листинга программы	12
3.7	Проверка работы изменённой программы	13
3.8	текст листинга в файле	14
3.9	Запуск программы с указанием аргументов	15
3.10	текст листинга в файле	16
3.11	запуск программы	16
3.12	Изменение листинга программы	17
3.13	запуск программы	17
3.14	Листинг данной программы	18
3.15	запуск программы	19

Список таблиц

1 Цель работы

Приобретение навыков написания программ с использованием циклов и обработкой аргументов командной строки. Целью работы является приобретение навыков работы с написанием программ с использованием циклов и обработкой аргументов командной строки. Использование полученных знаний на практике.

2 Теоретическое введение

8.2.1. 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. На рисунке 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).

8.2.1.1. 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.

8.2.1.2. 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.

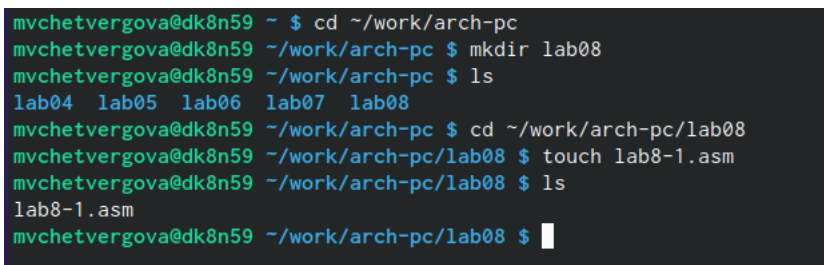
Аналогично команде записи в стек существует команда pusha, которая восстанавливает из стека все регистры общего назначения, и команда popf для перемещения значений из вершины стека в регистр флагов.

8.2.2. 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, типичная структура которого имеет следующий вид. 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.

3 Выполнение лабораторной работы

Реализация циклов в NASM

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



```
mvchetvergova@dk8n59 ~ $ cd ~/work/arch-pc
mvchetvergova@dk8n59 ~/work/arch-pc $ mkdir lab08
mvchetvergova@dk8n59 ~/work/arch-pc $ ls
lab04 lab05 lab06 lab07 lab08
mvchetvergova@dk8n59 ~/work/arch-pc $ cd ~/work/arch-pc/lab08
mvchetvergova@dk8n59 ~/work/arch-pc/lab08 $ touch lab8-1.asm
mvchetvergova@dk8n59 ~/work/arch-pc/lab08 $ ls
lab8-1.asm
mvchetvergova@dk8n59 ~/work/arch-pc/lab08 $
```

Рис. 3.1: Рис.1 Создание каталога и файла для работы

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). Листинг 8.1. Программа вывода значений регистра ecx

```
;----- ; Программа вывода значений регистра 'ecx';----- %include 'in_out.asm' SECTION .data msg1 db 'Введите N:',0h 88 Демидова А. В. Архитектура ЭВМ 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`

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

```
lab8-1.asm [----] 9 L: [ 1+30 31/ 31] *(849 / 849b) <EOF>
;-----
; Программа вывода значений регистра 'ecx'
;-----
%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. 3.2: Ввод листинга программы

```
mvchetvergova@dk8n59 ~/work/arch-pc/lab08 $ nasm -f elf lab8-1.asm
mvchetvergova@dk8n59 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-1 lab8-1.o
mvchetvergova@dk8n59 ~/work/arch-pc/lab08 $ ./lab8-1
Введите N: 6
6
5
4
3
2
1
```

Рис. 3.3: Проверка работы данной программы

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

```
label: sub ecx,1 ; ecx=ecx-1 mov [N],ecx mov eax,[N] call iprintLF
89 Архитектура ЭВМ loop label
```

Создайте исполняемый файл и проверьте его работу.

```

lab8-1.asm      [----] 9 L:[ 1+31 32/ 32] *(873 / 873b) <EOF>
;
; -----
; Программа вывода значений регистра 'ecx'
; -----
#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 ; 'ecx=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. 3.4: Изменение листинга программы

```

mvchetvergova@dk8n59 ~/work/arch-pc/lab08 $
mvchetvergova@dk8n59 ~/work/arch-pc/lab08 $
mvchetvergova@dk8n59 ~/work/arch-pc/lab08 $
mvchetvergova@dk8n59 ~/work/arch-pc/lab08 $ nasm -f elf lab8-1.asm
mvchetvergova@dk8n59 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-1 lab8-1.o
mvchetvergova@dk8n59 ~/work/arch-pc/lab08 $ ./lab8-1
Введите N: 6
5
3
1

```

Рис. 3.5: Проверка работы изменённой программы

Какие значения принимает регистр ecx в цикле? Соответствует ли число проходов цикла значению ☒ введенному с клавиатуры? Ответ: Регистр ecx принимает значение

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

`loop: label: push ecx ; добавление значения ecx в стек`
`sub ecx,1 mov [N],ecx mov`
`eax,[N] call iprintLF pop ecx ; извлечение значения ecx из стека`
`loop label`

```
lab8-1.asm          [----]  9 L: [ 1+34  35/ 35] *(891 / 891b) <EOF>
;
; Программа вывода значений регистра 'ecx'
;
%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:
push ecx
sub ecx,1 ; 'ecx=ecx-1'
mov [N],ecx
mov eax,[N]
call iprintLF ; Вывод значения 'N'
pop ecx

loop label ; 'ecx=ecx-1' и если 'ecx' не '0'
; ----- ; переход на 'label'
call quit
```

Рис. 3.6: Изменение листинга программы

```
mvchetvergova@dk8n59 ~/work/arch-pc/lab08 $ mcedit lab8-1.asm

mvchetvergova@dk8n59 ~/work/arch-pc/lab08 $ nasm -f elf lab8-1.asm
mvchetvergova@dk8n59 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-1 lab8-1.o
mvchetvergova@dk8n59 ~/work/arch-pc/lab08 $ ./lab8-1
Введите N: 6
5
4
3
2
1
0
```

Рис. 3.7: Проверка работы изменённой программы

Создайте исполняемый файл и проверьте его работу. Соответствует ли в данном случае число проходов цикла значению ☒ введенному с клавиатуры? Да, число проходов совпадает. Программа проводит 6 операций во время выполнения кода: работает с числами 6 с переходом в 0, 5 с переходом в 4, 4 с переходом в 3, 3 с переходом в 2, 2 с переходом в 1 и 1 с переходом в 0. После прохода с единицей получается 0 и программа прекращает действие.

8.3.2. Обработка аргументов командной строки

При разработке программ иногда встает необходимость указывать аргументы, которые будут использоваться в программе, непосредственно из командной строки при запуске программы.

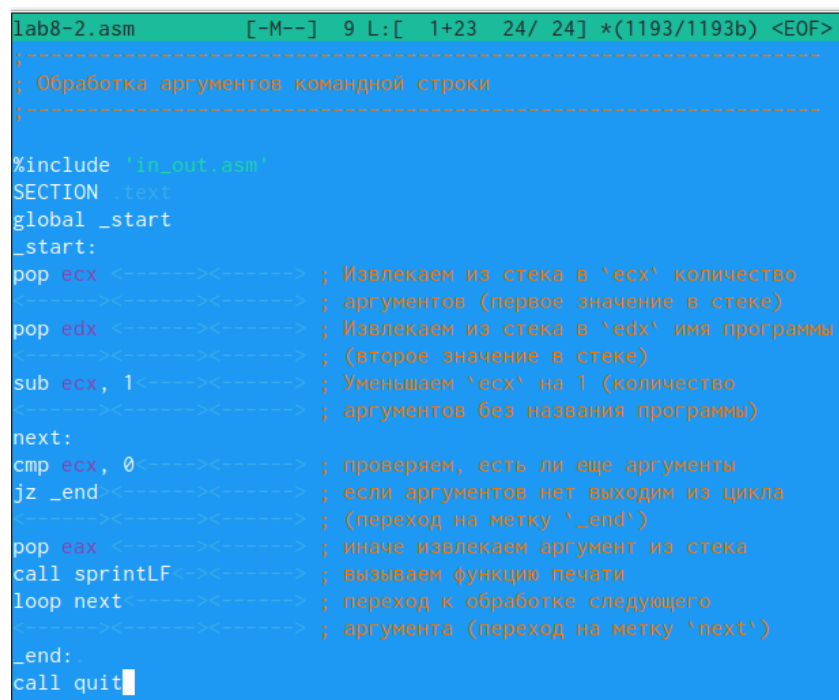
При запуске программы в NASM аргументы командной строки загружаются в стек в обратном порядке, кроме того в стек записывается имя программы и общее количество аргументов. Последние два элемента стека для программы, скомпилированной NASM, – это всегда имя программы и количество переданных аргументов.

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).

Листинг 8.2. Программа выводящая на экран аргументы командной строки

```
;----- ; Обработка аргументов командной строки ;-----  
%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
```

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



```
lab8-2.asm [-M--] 9 L: [ 1+23 24/ 24] *(1193/1193b) <EOF>  
;-----  
; Обработка аргументов командной строки  
;-----  
%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. 3.8: текст листинга в файле

Создайте исполняемый файл и запустите его, указав аргументы:

```

mvchetvergova@dk8n59 ~/work/arch-pc/lab08 $ nasm -f elf lab8-2.asm
mvchetvergova@dk8n59 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-2 lab8-2.o
mvchetvergova@dk8n59 ~/work/arch-pc/lab08 $ ./lab8-2 3 7 '4'
3
7
4

```

Рис. 3.9: Запуск программы с указанием аргументов

Сколько аргументов было обработано программой? Программа обработала 3 аргумента и вывела их на экран. Рассмотрим еще один пример программы которая выводит сумму чисел, которые передаются в программу как аргументы. Создайте файл lab8-3.asm в каталоге и введите в него текст программы из листинга 8.3.

Листинг 8.3. Программа вычисления суммы аргументов командной строки

```

%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 аргумен-
тов 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 ; пе-
реход 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

```

Создайте исполняемый файл и запустите его, указав аргументы. Пример результата работы программы:

```

lab8-3.asm      [-M--] 46 L: [ 1+28 29/ 29] *(1477/1477b) <EOF>
#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. 3.10: текст листинга в файле

```

mvchetvergova@dk8n59 ~/work/arch-pc/lab08 $ mcedit lab8-3.asm
mvchetvergova@dk8n59 ~/work/arch-pc/lab08 $ nasm -f elf lab8-3.asm
mvchetvergova@dk8n59 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-3 lab8-3.o
mvchetvergova@dk8n59 ~/work/arch-pc/lab08 $ ./lab8-3 5 15 48 4
Результат: 72

```

Рис. 3.11: запуск программы

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


```

lab8-3.asm      [----]  0 L:[ 1+25 26/ 35] *(1090/144
#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, 1 ; 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
mul esi ; добавляем к промежуточной сумме
; след. аргумент 'esi=esi+eax'
mov 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. 3.12: Изменение листинга программы

```

mvchetvergova@dk4n60 ~$ cd ~/work/study/2023-2024/Архитектура
mvchetvergova@dk4n60 ~$ cd ~/work/arch-pc/lab08
mvchetvergova@dk4n60 ~/work/arch-pc/lab08 $ ./lab8-3 5 5 5 5
Результат: 625
mvchetvergova@dk4n60 ~/work/arch-pc/lab08 $

```

Рис. 3.13: запуск программы

8.4. 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 No 7. Создайте исполняемый файл и проверьте его работу на нескольких наборах $x = x_1 + x_2 + \dots + x_n$.

```
lab8-4.asm [----] 13 L: [ 1+ 2 3/ 39] *(49 /1536b) 1056 0x420
#include "in_out.asm"
SECTION .data
msg1 db "Результат: ",0h
msg2 db "Функция: f(x)=3(x+2)",0h
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 eax,2
mov ebx, 3
mul ebx
add esi,eax ; добавляем к промежуточной сумме
; след. аргумент 'esi=esi+eax'
loop next ; переход к обработке следующего аргумента

_end:
mov eax, msg2
call sprint
mov eax, msg1 ; вывод сообщения "Результат: "
call sprint
mov eax, esi ; записываем сумму в регистр 'eax'
call iprintfLF ; печать результата
call quit ; завершение программы
```

Рис. 3.14: Листинг данной программы

```
mvchetvergova@dk4n60 ~/work/arch-pc/lab08 $ nasm -f elf lab8-4.asm
mvchetvergova@dk4n60 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-4 lab8-4.o
mvchetvergova@dk4n60 ~/work/arch-pc/lab08 $ ./lab8-4 2 3 4
Функция: f(x)=3(x+2)    Результат: 45
```

Рис. 3.15: запуск программы

4 Выводы

В ходе выполнения лабораторной работы номер 8, мы приобрели навыки написания программ с использованием циклов и обработкой аргументов командной строки. Целью работы является приобретение навыков работы с написанием программ с использованием циклов и обработкой аргументов командной строки. Использовали полученные знания на практике.