

Лабораторная работа №8

Программирование цикла. Обработка аргументов командной строки

Андрюшин Никита Сергеевич

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
3	Выполнение задания для самостоятельной работы	16
4	Выводы	19

List of Figures

2.1	Создание рабочей директории и файла lab8-1.asm	6
2.2	Запуск Midnight commander	6
2.3	Вставка кода из файла листинга 8.1	7
2.4	Копирование файла in_out.asm в рабочую директорию	8
2.5	Сборка программы из файла lab8-1.asm и её запуск	8
2.6	Изменение файла lab8-1.asm	9
2.7	Повторная сборка программы из файла lab8-1.asm и её запуск . .	9
2.8	Результат вывода	10
2.9	Результат вывода для чётного N	10
2.10	Редактирование файла lab8-1.asm	11
2.11	Повторная сборка программы из файла lab8-1.asm и её запуск . .	11
2.12	Создание второго файла: lab8-2.asm	12
2.13	Запись кода из листинга 8.2 в файл lab8-2.asm	12
2.14	Сборка программы из файла lab8-2.asm и её запуск	13
2.15	Создание третьего файла: lab8-3.asm	13
2.16	Запись кода из листинга 8.3 в файл lab8-3.asm	14
2.17	Сборка программы из файла lab8-2.asm и её запуск	14
2.18	Изменение файла lab8-3.asm	15
2.19	Повторная сборка программы из файла lab8-3.asm и её запуск . .	15
3.1	Создание файла самостоятельной работы	16
3.2	Код файла самостоятельной работы	17
3.3	Сборка и запуск программы первого задания самостоятельной ра- боты, а также результат выполнения	18

List of Tables

1 Цель работы

Научиться работать с циклами на языке Ассемблера, а также научиться обрабатывать аргументы командной строки

2 Выполнение лабораторной работы

Для начала выполнения лабораторной работы создадим рабочую директорию и файл lab8-1.asm :

```
rapavlichenko@rapavlichenko:~$ mkdir ~/work/arch-pc/lab08
rapavlichenko@rapavlichenko:~$ cd ~/work/arch-pc/lab08
rapavlichenko@rapavlichenko:~/work/arch-pc/lab08$ touch lab8-1.asm
rapavlichenko@rapavlichenko:~/work/arch-pc/lab08$
```

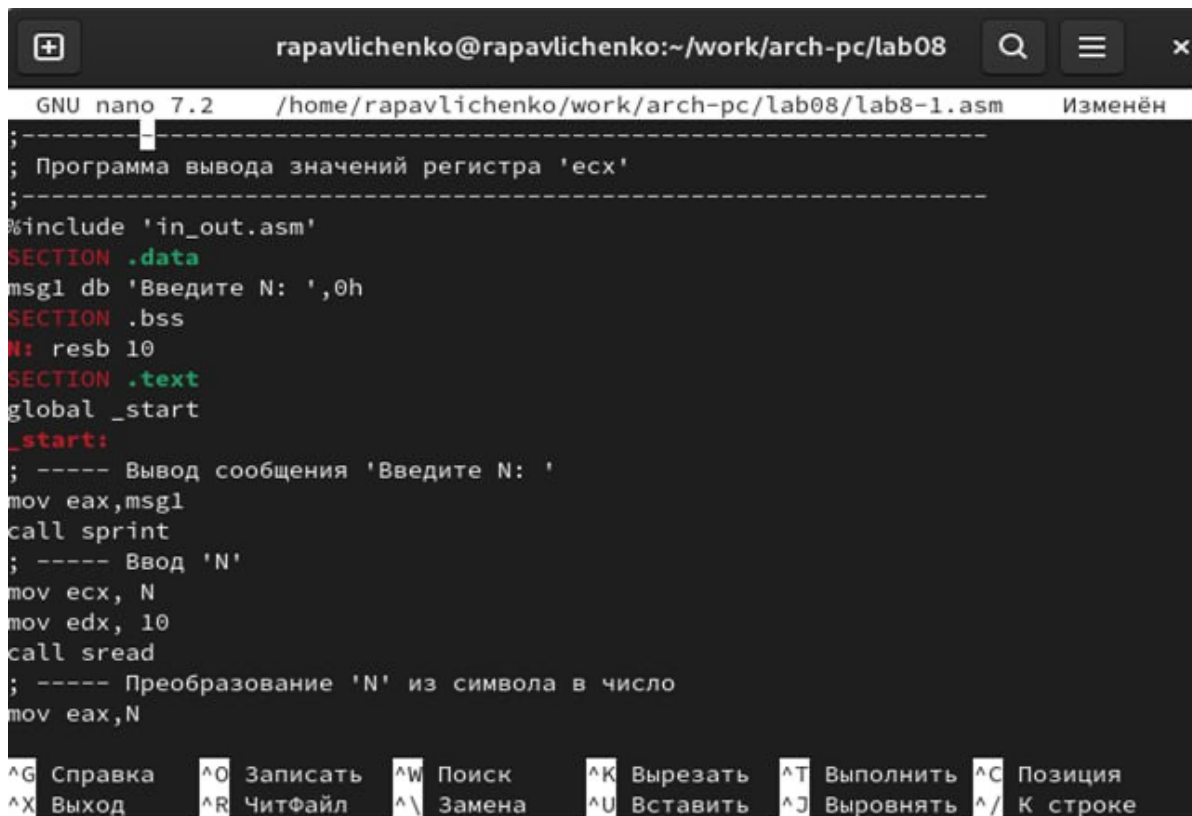
Figure 2.1: Создание рабочей директории и файла lab8-1.asm

Далее, запустим Midnight commander :

```
rapavlichenko@rapavlichenko:~/work/arch-pc/lab08$ mc
```

Figure 2.2: Запуск Midnight commander

Теперь, вставим в ранее созданный файл из листинга 8.1. Он должен запускать цикл и выводить каждую итерацию число, на единицу меньше предыдущего (начинается выводить с числа N) :



```
rapavlichenko@rapavlichenko:~/work/arch-pc/lab08
GNU nano 7.2 /home/rapavlichenko/work/arch-pc/lab08/lab8-1.asm  Изменён
;-----
; 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
^G Справка  ^O Записать  ^W Поиск     ^K Вырезать  ^T Выполнить  ^C Позиция
^X Выход    ^R ЧитФайл  ^\ Замена   ^U Вставить  ^J Выровнять  ^/ К строке
```

Figure 2.3: Вставка кода из файла листинга 8.1

Чтобы собрать код, нужен файл `in_out.asm`. скопируем его из директории прошлой лабораторной работы :

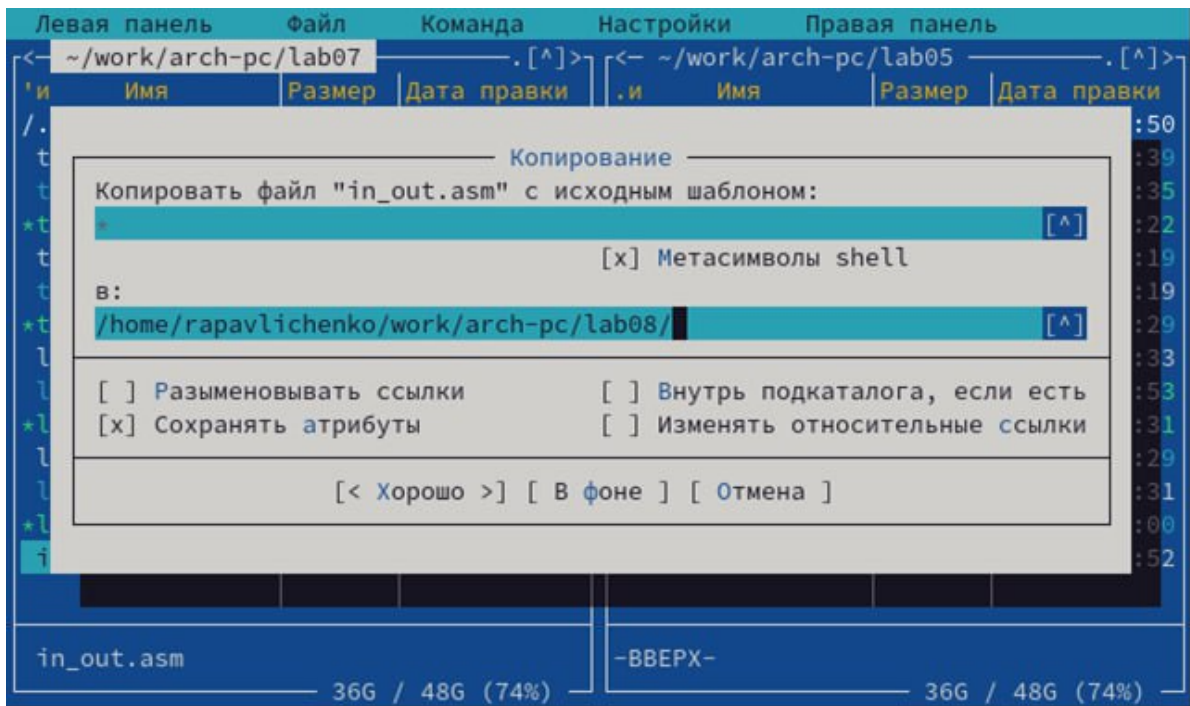


Figure 2.4: Копирование файла in_out.asm в рабочую директорию

Теперь соберём программу и посмотрим на результат выполнения :

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

Figure 2.5: Сборка программы из файла lab8-1.asm и её запуск

Как видим, она выводит числа от N до единицы включительно. Теперь попробуем изменить код, чтобы в цикле также отнималась единица у регистра ecx :


```

GNU nano 7.2 /home/rapavlichenko/work/arch-pc/lab08/lab8-1.asm Изменён
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

```

Figure 2.6: Изменение файла lab8-1.asm

Попробуем собрать программу и запустить её :

```

rapavlichenko@rapavlichenko:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
rapavlichenko@rapavlichenko:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
rapavlichenko@rapavlichenko:~/work/arch-pc/lab08$ ./lab8-1

```

Figure 2.7: Повторная сборка программы из файла lab8-1.asm и её запуск

Введём в качестве N число 5 и посмотрим на результат выполнения :

```

4294687464
4294687462
4294687460
4294687458
4294687456
4294687454
4294687452
4294687450
4294687448
4294687446
4294687444
4294687442
4294687440
4294687438
4294687436
4294687434
4294687432
4294687430
4294687428
4294687426
4294687424
4294687422
4294687420
4294687418
4294687416
4294687414
429468741^Z
[3]+ Остановлен ./lab8-1
rapavlichenko@rapavlichenko:~/work/arch-pc/lab08$

```

Figure 2.8: Результат вывода

Как видим, цикл выполняется бесконечное количество раз. Это связано с тем, что цикл останавливается в тот момент, когда при проверке `esx` равен 0, но он каждое выполнение цикла уменьшается на 2, из-за чего, в случае нечётного числа, никогда не достигнет нуля. Регистр `esx` меняет своё значение дважды: стандартно -1 после каждой итерации и -1 в теле цикла из-за команды `sub`. Если на вход подать чётное число, цикл прогонится $N/2$ раз, выводя числа от $N-1$ до 1 (выводит через одно число) :

```

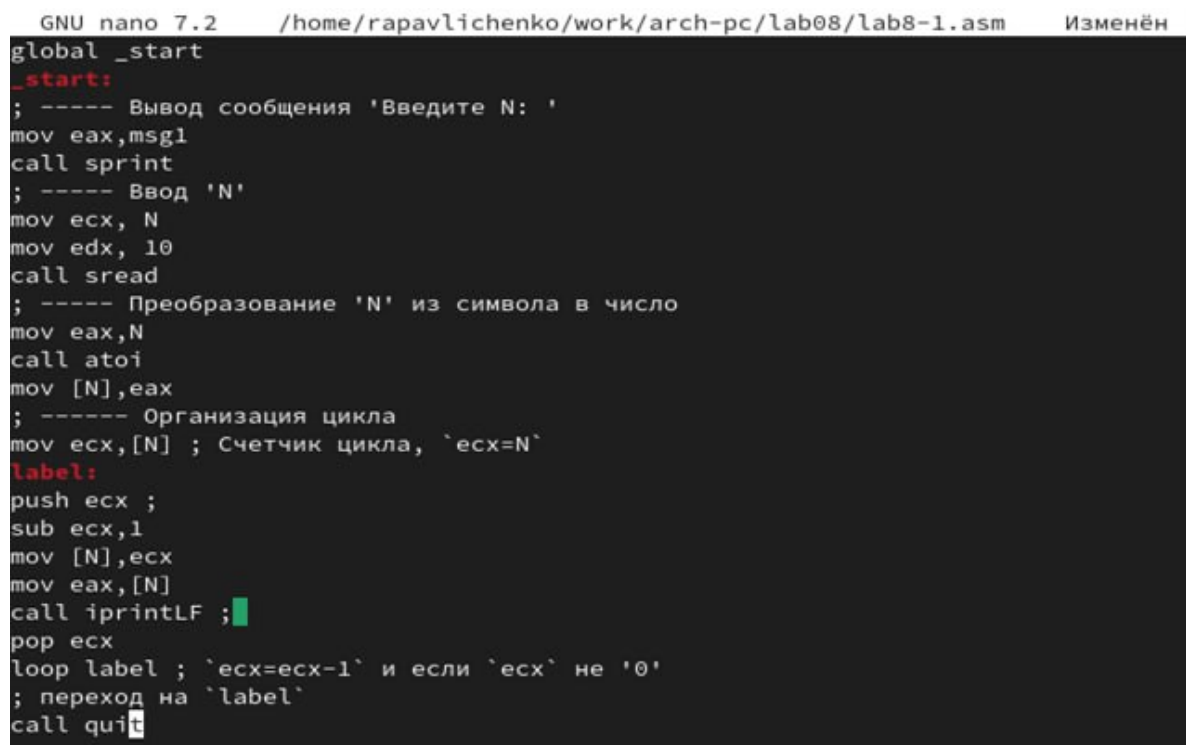
rapavlichenko@rapavlichenko:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 12
11
9
7
5
3
1
rapavlichenko@rapavlichenko:~/work/arch-pc/lab08$

```

Figure 2.9: Результат вывода для чётного N

Таким образом, количество итераций цикла не равно N ни при подаче на вход чётного числа, ни при подаче нечётного.

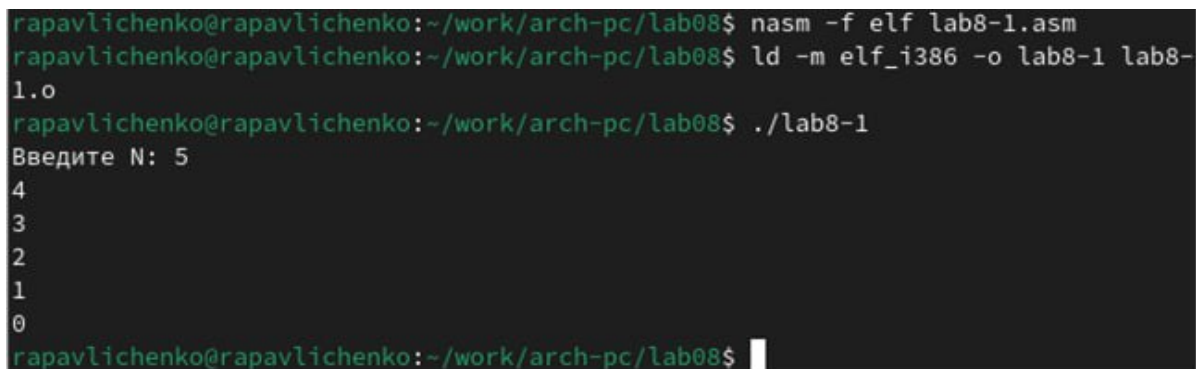
Теперь попробуем изменить программу так, чтобы она сохраняла значение регистра `ecx` в стек :



```
GNU nano 7.2 /home/rapavlichenko/work/arch-pc/lab08/lab8-1.asm Изменён
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
mov [N],ecx
mov eax,[N]
call iprintLF ;
pop ecx
loop label ; `ecx=ecx-1` и если `ecx` не '0'
; переход на `label`
call quit
```

Figure 2.10: Редактирование файла `lab8-1.asm`

Попробуем собрать и запустить программу :



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

Figure 2.11: Повторная сборка программы из файла `lab8-1.asm` и её запуск

Теперь, программа выводит все числа от `N-1` до нуля. Таким образом, число прогонов цикла равно числу `N`. Создадим второй файл :

```
rapavlichenko@rapavlichenko:~/work/arch-pc/lab08$ touch lab8-2.asm
rapavlichenko@rapavlichenko:~/work/arch-pc/lab08$
```

Figure 2.12: Создание второго файла: lab8-2.asm

И вставим в него код из файла листинга 8.2 :

```
GNU nano 7.2 /home/rapavlichenko/work/arch-pc/lab08/lab8-2.asm Изменён
;-----
; 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 sprintf ; вызываем функцию печати
loop next ; переход к обработке следующего
; аргумента (переход на метку `next`)
_end:
call quit
```

Figure 2.13: Запись кода из листинга 8.2 в файл lab8-2.asm

Соберём и запустим его, указав некоторые аргументы. Посмотрим на результат :

```

rapavlichenko@rapavlichenko:~/work/arch-pc/lab08$ nasm -f elf lab8-2.asm
rapavlichenko@rapavlichenko:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-2 lab8-2.o
rapavlichenko@rapavlichenko:~/work/arch-pc/lab08$ ./lab8-2
rapavlichenko@rapavlichenko:~/work/arch-pc/lab08$ ./lab8-2 аргумент1 аргумент 2
'аргумент 3'
аргумент1
аргумент
2
аргумент 3
rapavlichenko@rapavlichenko:~/work/arch-pc/lab08$

```

Figure 2.14: Сборка программы из файла lab8-2.asm и её запуск

Как видим, он обработал 4 аргумента. Аргументы разделяются пробелом, либо, когда аргумент содержит в себе пробел, обрамляется в кавычки. Создадим третий файл :

```

rapavlichenko@rapavlichenko:~/work/arch-pc/lab08$ touch lab8-3.asm
rapavlichenko@rapavlichenko:~/work/arch-pc/lab08$

```

Figure 2.15: Создание третьего файла: lab8-3.asm

И вставим в него код из листинга 8.3. Он будет находить сумму всех аргументов :

```

GNU nano 7.2 /home/rapavlichenko/work/arch-pc/lab08/lab8-3.asm Изменён
#include 'in_out.asm'
SECTION .data
msg db "Результат: ",0
SECTION .text
global _start
_start:
Архитектура ЭВМ
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:

```

Figure 2.16: Запись кода из листинга 8.3 в файл lab8-3.asm

Теперь соберём программу и запустим её :

```

rapavlichenko@rapavlichenko:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
rapavlichenko@rapavlichenko:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
rapavlichenko@rapavlichenko:~/work/arch-pc/lab08$ ./lab8-3 13 10 18 3 5
Результат: 49
rapavlichenko@rapavlichenko:~/work/arch-pc/lab08$

```

Figure 2.17: Сборка программы из файла lab8-2.asm и её запуск

Как видим, программа действительно выводит сумму всех аргументов. Изменим её так, чтобы она находила не сумму, а произведение всех аргументов :


```

GNU nano 7.2 /home/rapavlichenko/work/arch-pc/lab08/lab8-3.asm Изменён
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
mul esi
mov esi,eax
loop next ; переход к обработке следующего аргумента
_end:
mov ebx, eax
mov eax, msg ; вывод сообщения "Результат: "
call sprint
mov eax, ebx ; записываем сумму в регистр `eax`
call iprintLF ; печать результата
call quit ; завершение программы

```

Figure 2.18: Изменение файла lab8-3.asm

Соберём программу и запустим её :

```

rapavlichenko@rapavlichenko:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
rapavlichenko@rapavlichenko:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
rapavlichenko@rapavlichenko:~/work/arch-pc/lab08$ ./lab8-3 2 3 4 5
Результат: 120
rapavlichenko@rapavlichenko:~/work/arch-pc/lab08$ ./lab8-3 4 5 6 7
Результат: 840
rapavlichenko@rapavlichenko:~/work/arch-pc/lab08$


```

Figure 2.19: Повторная сборка программы из файла lab8-3.asm и её запуск

Как видим, программа выводит правильный ответ

3 Выполнение задания для самостоятельной работы

Для выполнения самостоятельной работы создадим файл в формате .asm :



```
rapavlichenko@rapavlichenko:~/work/arch-pc/lab08$ touch task1v1.asm
rapavlichenko@rapavlichenko:~/work/arch-pc/lab08$
```

Figure 3.1: Создание файла самостоятельной работы

В рамках самостоятельной работы необходимо сделать задание под вариантом 9. Так, необходимо сложить результаты выполнения функции $f(x)=2x+15$ для всех введённых аргументов :


```

GNU nano 7.2 /home/rapavlichenko/work/arch-pc/lab08/task1v1.asm Изменён
%include 'in_out.asm'
SECTION .data
msg db "Результат: ", 0
msg2 db "Функция: f(x)=2x+15", 0
SECTION .text
global _start
_start:
pop ecx          ; Извлекаем из стека количество аргументов
pop edx          ; Извлекаем из стека имя программы
sub ecx, 1       ; Уменьшаем ecx на 1 (только аргументы, без имени программы)
mov esi, 0       ; Используем esi для хранения промежуточной суммы
next:
cmp ecx, 0       ; Проверяем, остались ли аргументы
jz _end          ; Если нет, переходим к завершению программы
pop eax          ; Извлекаем следующий аргумент из стека
call atoi        ; Преобразуем строку в число
mov ebx, 2       ; Умножаем на 2
mul ebx
add eax, 15      ; Добавляем 15
add esi, eax     ; Суммируем результат с текущей суммой
loop next        ; Переходим к обработке следующего аргумента
_end:
mov eax, msg2    ; Выводим сообщение с функцией
call sprintf
mov eax, msg     ; Выводим сообщение "Результат: "
call sprintf
mov eax, esi     ; Записываем сумму в регистр eax
call iprintLF    ; Выводим результат
call quit        ; Завершаем программу

```

Figure 3.2: Код файла самостоятельной работы

Соберём и запустим программу, вводя различные аргументы :

```
rapavlichenko@rapavlichenko:~/work/arch-pc/lab08$ nasm -f elf tasklv1.asm
rapavlichenko@rapavlichenko:~/work/arch-pc/lab08$ ld -m elf_i386 -o tasklv1 tasklv1.o
rapavlichenko@rapavlichenko:~/work/arch-pc/lab08$ ./tasklv1
Функция:  $f(x)=2x+15$ 
Результат: 0
rapavlichenko@rapavlichenko:~/work/arch-pc/lab08$ ./tasklv1 1 2 3 4
Функция:  $f(x)=2x+15$ 
Результат: 80
rapavlichenko@rapavlichenko:~/work/arch-pc/lab08$ ./tasklv1 5 6 7 8
Функция:  $f(x)=2x+15$ 
Результат: 112
rapavlichenko@rapavlichenko:~/work/arch-pc/lab08$ ./tasklv1 10 11 12 13 14 15
Функция:  $f(x)=2x+15$ 
Результат: 240
rapavlichenko@rapavlichenko:~/work/arch-pc/lab08$
```

Figure 3.3: Сборка и запуск программы первого задания самостоятельной работы, а также результат выполнения

Пересчитав результат вручную, убеждаемся, что программа работает верно

4 Выводы

В результате выполнения лабораторной работы были получены навыки работы с циклами и обработкой аргументов из командной строки. Были написаны программы, использующие все вышеописанные аспекты