

Лабораторная работа №8.

Программирование цикла.Обработка аргументов командной строки.

Швед Карина Дмитриевна

Содержание

1	Цель работы	5
2	Ход работы	6
3	Задание для самостоятельной работы	14
4	Выводы	16

Список иллюстраций

2.1	Код программы в файле lab8-1.asm	7
2.2	Работа программы lab8-1.asm	7
2.3	Код программы в файле lab8-1.asm	8
2.4	Запуск программы lab8-1.asm	9
2.5	Запуск программы lab8-1.asm	9
2.6	Код программы lab8-1.asm	10
2.7	Работа программы lab8-1.asm	10
2.8	Код программы lab8-2.asm	11
2.9	Работа программы lab8-2.asm	11
2.10	Код программы lab8-3.asm	12
2.11	Работа программы lab8-3.asm	12
2.12	Код программы lab8-3.asm	13
2.13	Работа программы lab8-3.asm	13
3.1	Код программы task8-1.asm	15
3.2	Работа программы task8-1.asm	15

Список таблиц

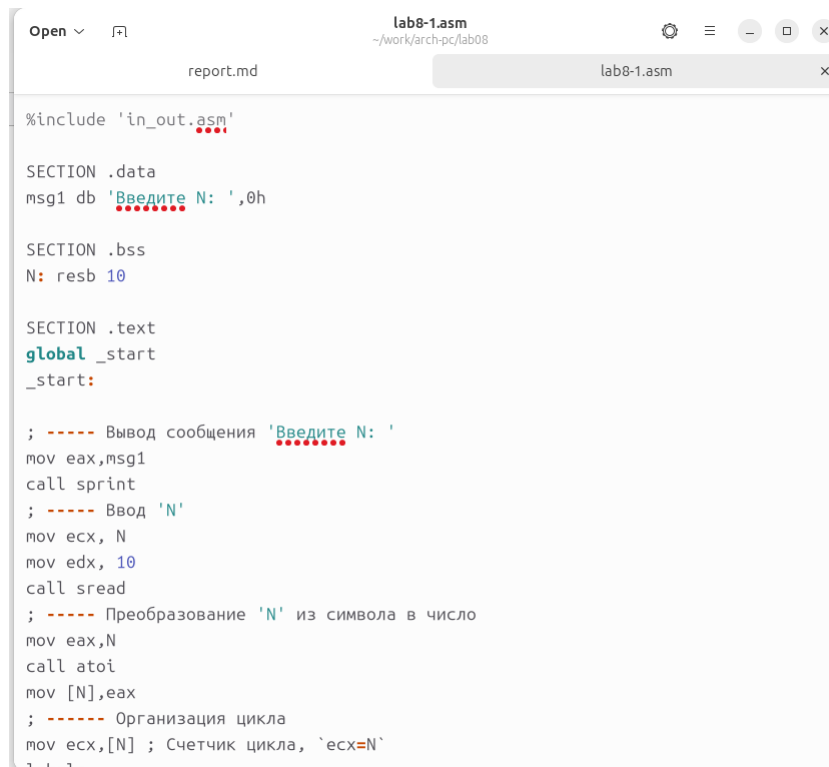
1 Цель работы

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

2 Ход 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`.

Создала каталог для программ лабораторной работы № 8, перешла в него и создала файл `lab8-1.asm`. Далее я внимательно изучила текст программы из листинга 8.1 и ввела в файл `lab8-1.asm`. (рис. 2.1). Создала исполняемый файл и проверила его работу.(рис. 2.2).



```
Open  report.md  lab8-1.asm  ~/work/arch-pc/lab08  lab8-1.asm  x

%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:
```

Рис. 2.1: Код программы в файле lab8-1.asm

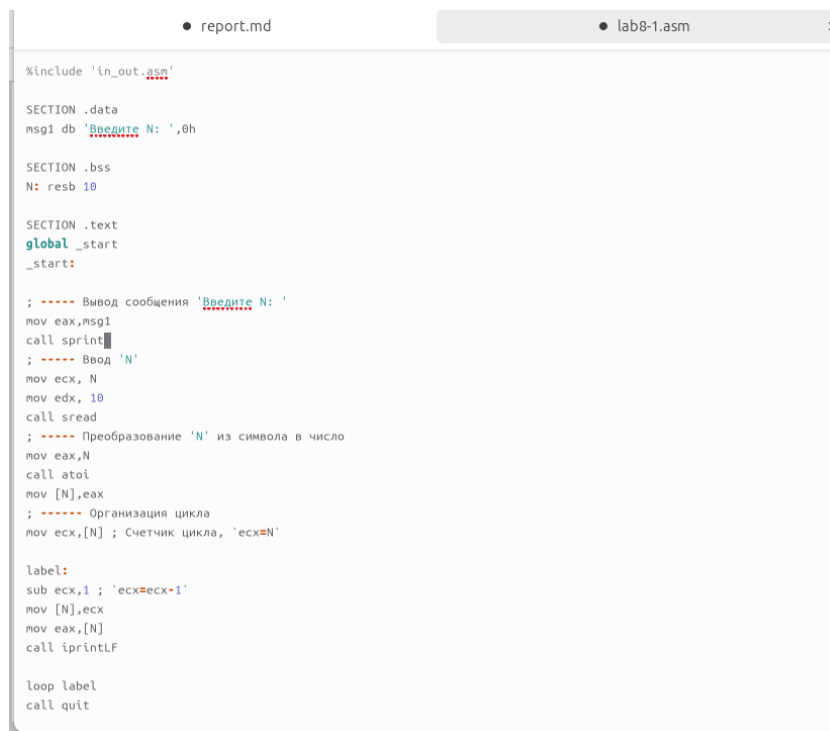


```
vboxuser@hiy: ~/work/arch-pc/lab08
vboxuser@hiy:~$ mkdir ~/work/arch-pc/lab08
vboxuser@hiy:~$ cd ~/work/arch-pc/lab08
vboxuser@hiy:~/work/arch-pc/lab08$ touch lab8-1.asm
vboxuser@hiy:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
vboxuser@hiy:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
vboxuser@hiy:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 5
5
4
3
2
1
vboxuser@hiy:~/work/arch-pc/lab08$ 9
9: command not found
vboxuser@hiy:~/work/arch-pc/lab08$
```

Рис. 2.2: Работа программы lab8-1.asm

Данный пример показывает, что использование регистра ecx в теле цикла loop может привести к некорректной работе программы. Я изменила текст программ,

добавив изменение значение регистра ecx в цикле (рис. 2.3). Создала исполняемый файл и проверила его работу.



```
%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

loop label
call quit
```

Рис. 2.3: Код программы в файле lab8-1.asm

При нечетном N программа запускает бесконечный цикл (рис. 2.4), а при четном N выводит только нечетные числа (рис. 2.5)

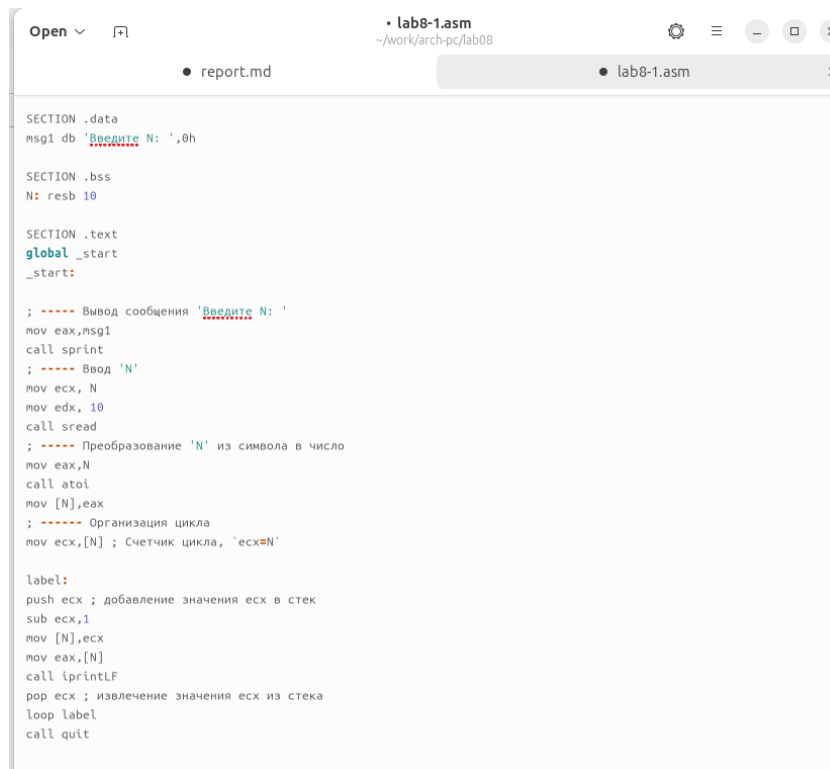

```
vboxuser@hiy: ~/work/arch-pc/lab08
4294964290
4294964288
4294964286
4294964284
4294964282
4294964280
4294964278
4294964276
4294964274
4294964272
4294964270
4294964268
4294964266
4294964264
4294964262
4294964260
4294964258
4294964256
4294964254
4294964252
4294964250
4294964248
4294964^C
```

Рис. 2.4: Запуск программы lab8-1.asm

```
vboxuser@hiy: ~/work/arch-pc/lab08$ ./lab8-1
Введите N: 2
1
vboxuser@hiy: ~/work/arch-pc/lab08$ ./lab8-1
Введите N: 6
5
3
1
vboxuser@hiy: ~/work/arch-pc/lab08$
```

Рис. 2.5: Запуск программы lab8-1.asm

Для использования регистра `ecx` в цикле и сохранения корректности работы программы можно использовать стек. Я добавила в программу команды `push` и `pop` (добавления в стек и извлечения из стека) для сохранения значения счетчика цикла `loop` (рис. 2.6) Создала исполняемый файл и проверила его работу (рис. 2.7) Программа выводит числа от $N-1$ до 0, число проходов цикла соответствует N .



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

Рис. 2.6: Код программы lab8-1.asm



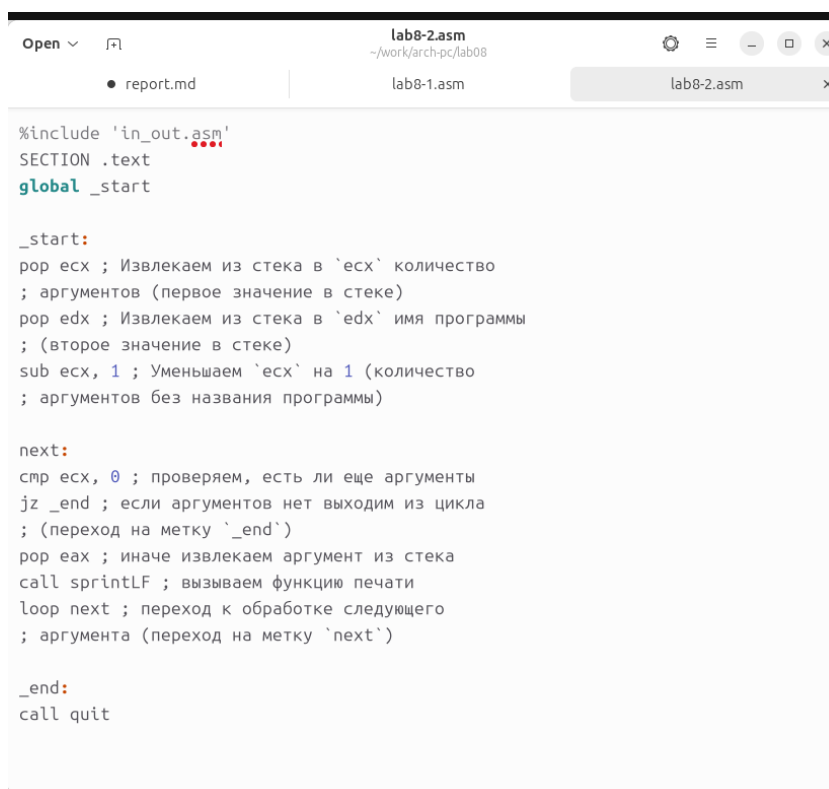
```
vboxuser@hiy: ~/work/arch-pc/lab08
vboxuser@hiy:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 2
1
vboxuser@hiy:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 6
5
3
1
vboxuser@hiy:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
vboxuser@hiy:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
vboxuser@hiy:~/work/arch-pc/lab08$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

Рис. 2.7: Работа программы lab8-1.asm

При разработке программ иногда встает необходимость указывать аргументы, которые будут использоваться в программе, непосредственно из командной стро-

ки при запуске программы. Для того чтобы использовать аргументы в программе, их просто нужно извлечь из стека. Обработку аргументов нужно проводить в цикле. В качестве примера рассмотрим программу, которая выводит на экран аргументы командной строки. Я внимательно изучила текст программы из Листинга 8.2.

Создала файл lab8-2.asm в каталоге ~/work/arch-pc/lab08 и ввела в него текст программы из листинга 8.2. (рис. 2.8) Далее создала исполняемый файл и запустила его, указав аргументы (рис. 2.9). Было обработано 4 аргумента




```
%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 printf ; вызываем функцию печати
        loop next ; переход к обработке следующего
                 ; аргумента (переход на метку `next`)

    _end:
        call quit
```

Рис. 2.8: Код программы lab8-2.asm



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

Рис. 2.9: Работа программы lab8-2.asm

Рассмотрим еще один пример программы которая выводит сумму чисел, которые передаются в программу как аргументы. Я создала файл lab8-3.asm в каталоге ~/work/arch-pc/lab08 и ввела в него текст программы из листинга 8.3. (рис. 2.10) Создала исполняемый файл и запустила его, указав аргументы. (рис. 2.11)

```

%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:
    shr 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 iprintf ; печать результата
    call quit ; завершение программы
  
```

Рис. 2.10: Код программы lab8-3.asm

```

vboxuser@hiy:~/work/arch-pc/lab08$ touch lab8-3.asm
vboxuser@hiy:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
vboxuser@hiy:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
vboxuser@hiy:~/work/arch-pc/lab08$ ./lab8-3 1 2 3 6
Результат: 12
vboxuser@hiy:~/work/arch-pc/lab08$
  
```

Рис. 2.11: Работа программы lab8-3.asm

Изменила текст программы из листинга 8.3 для вычисления произведения аргументов командной строки.(рис. 2.12) (рис. 2.13)

```
Open ▾ [icon] lab8-3.asm
~/work/arch-pc/lab08
• report.md lab8-3.asm x

%include 'in_out.asm'

SECTION .data
msg db "Результат: ",0

SECTION .text
global _start
_start:
    pop ecx          ; Извлекаем количество аргументов в `ecx`
    pop edx          ; Извлекаем имя программы в `edx`
    sub ecx, 1       ; Уменьшаем `ecx` на 1 (количество аргументов без названия программы)
    mov esi, 1       ; Устанавливаем начальное значение для произведения (1)

next:
    cmp ecx, 0       ; Проверяем, есть ли ещё аргументы
    jz _end          ; Если аргументов больше нет, выходим из цикла
    pop eax          ; Иначе извлекаем следующий аргумент из стека
    call atoi        ; Преобразуем аргумент из строки в число
    imul esi, eax    ; Умножаем текущий результат на аргумент (esi = esi * eax)
    loop next        ; Переход к обработке следующего аргумента

_end:
    mov eax, msg     ; Вывод сообщения "Результат: "
    call sprint
    mov eax, esi     ; Записываем произведение в регистр `eax`
    call iprintLF    ; Печать результата
    call quit        ; Завершение программы
```

Рис. 2.12: Код программы lab8-3.asm

```
vboxuser@hiy:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
vboxuser@hiy:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
vboxuser@hiy:~/work/arch-pc/lab08$ ./lab8-3 1 2 3 6
Результат: 36
vboxuser@hiy:~/work/arch-pc/lab08$
```

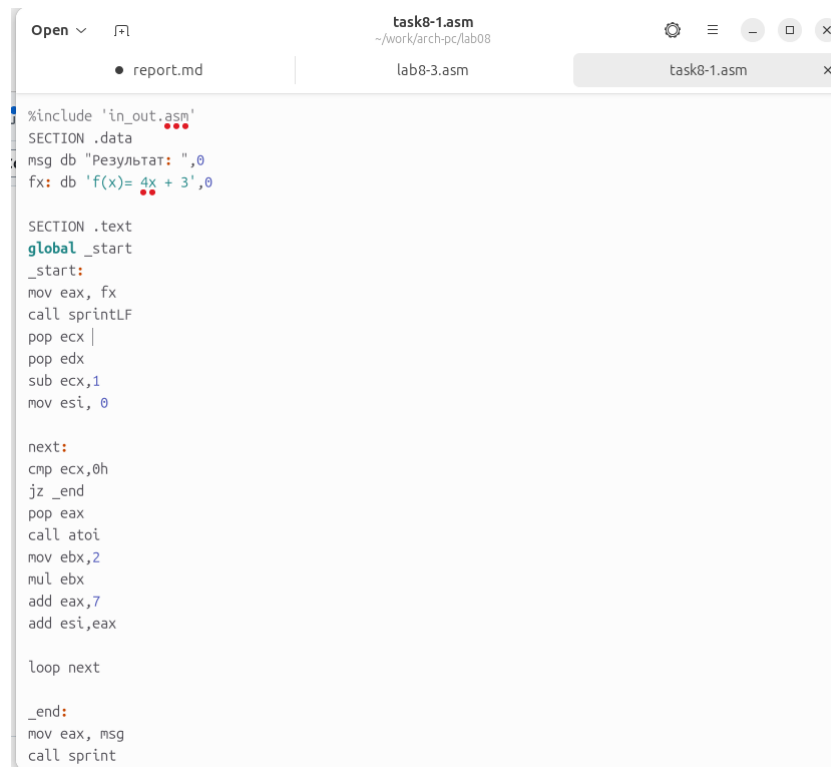
Рис. 2.13: Работа программы lab8-3.asm

3 Задание 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_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 $x = x_1, x_2, \dots, x_n$.

Для варианта 5: $4x + 3$

Сначала я создала файл task8-1.asm. Затем ввела соответствующий код. (рис. 3.1) Создала исполняемый файл и проверила работу файла. Я ввела несколько аргументов и получила сумму значений (рис. 3.2)



```
Open ▾  task8-1.asm
~\work\arch-pc\lab08
report.md  lab8-3.asm  task8-1.asm x

%include 'in_out.asm'
SECTION .data
msg db "Результат: ",0
fx: db 'f(x)= 4x + 3',0

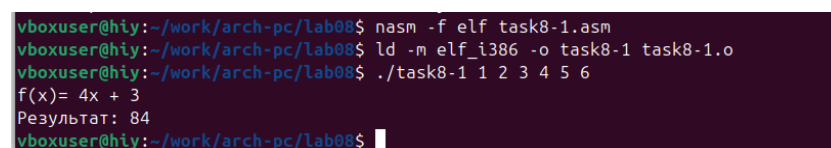
SECTION .text
global _start
_start:
mov eax, fx
call sprintf
pop ecx
pop edx
sub ecx,1
mov esi, 0

next:
cmp ecx,0h
jz _end
pop eax
call atoi
mov ebx,2
mul ebx
add eax,7
add esi,eax

loop next

_end:
mov eax, msg
call sprint
```

Рис. 3.1: Код программы task8-1.asm



```
vboxuser@hiy:~/work/arch-pc/lab08$ nasm -f elf task8-1.asm
vboxuser@hiy:~/work/arch-pc/lab08$ ld -m elf_i386 -o task8-1 task8-1.o
vboxuser@hiy:~/work/arch-pc/lab08$ ./task8-1 1 2 3 4 5 6
f(x)= 4x + 3
Результат: 84
vboxuser@hiy:~/work/arch-pc/lab08$
```

Рис. 3.2: Работа программы task8-1.asm

4 Выводы

Я приобрела навыки написания программ с использованием циклов и обработкой аргументов командной строки.