

Отчёт по лабораторной работе 8

Программирование цикла. Обработка аргументов командной строки.

Зиборова Вероника Николаевна НММбд-02-24

Содержание

1	Цель работы	5
2	Теоретическое введение	6
3	Выполнение лабораторной работы	7
3.1	Реализация циклов в NASM	7
3.2	Обработка аргументов командной строки	13
3.3	Задание для самостоятельной работы	17
4	Выводы	20
5	Ответы на вопросы	21

Список иллюстраций

3.1	Программа в файле lab8-1.asm	8
3.2	Запуск программы lab8-1.asm	9
3.3	Программа в файле lab8-1.asm	10
3.4	Запуск программы lab8-1.asm	11
3.5	Программа в файле lab8-1.asm	12
3.6	Запуск программы lab8-1.asm	13
3.7	Программа в файле lab8-2.asm	14
3.8	Запуск программы lab8-2.asm	14
3.9	Программа в файле lab8-3.asm	15
3.10	Запуск программы lab8-3.asm	15
3.11	Программа в файле lab8-3.asm	16
3.12	Запуск программы lab8-3.asm	17
3.13	Программа в файле task.asm	18
3.14	Запуск программы task.asm	19

Список таблиц

1 Цель работы

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

2 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.

Для стека существует две основные операции:

- добавление элемента в вершину стека (push);
- извлечение элемента из вершины стека (pop).

3 Выполнение лабораторной работы

3.1 Реализация циклов в NASM

Создала каталог для программ лабораторной работы № 8 и файл lab8-1.asm.

При реализации циклов в NASM с использованием инструкции `loop` необходимо помнить, что эта инструкция использует регистр `ecx` в качестве счетчика, уменьшая его значение на единицу с каждым шагом. В качестве примера я рассмотрела программу, которая выводит текущее значение регистра `ecx`.

Добавила в файл lab8-1.asm текст программы из листинга 8.1. Затем создала исполняемый файл и проверила его работу.

```
lab8-1.asm [----] 11 L: [ 1+22 23/ 28] *(484 / 636b) 0010 0x00A
#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
```

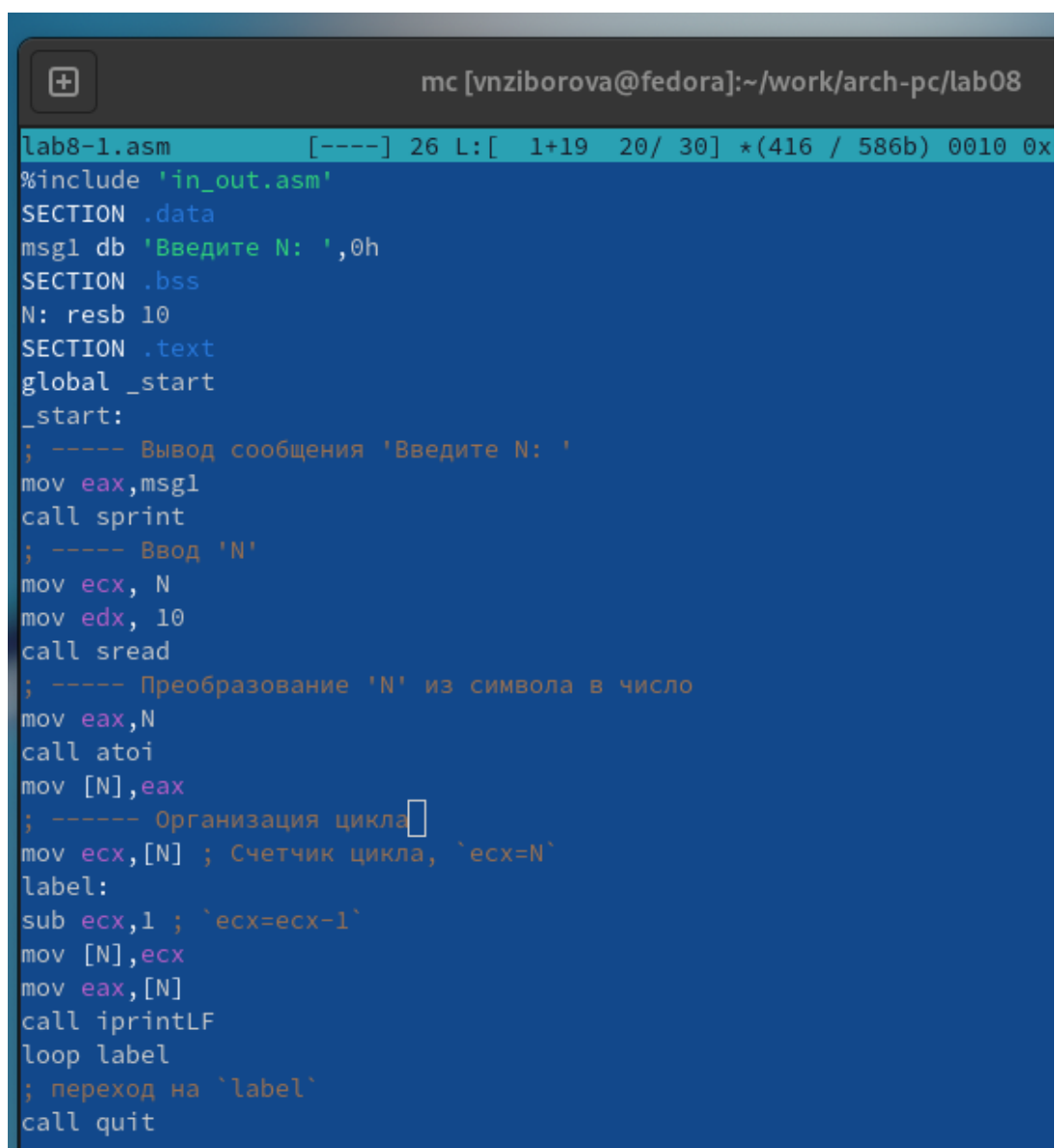
Рис. 3.1: Программа в файле lab8-1.asm


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

Рис. 3.2: Запуск программы lab8-1.asm

Этот пример показал, что изменение регистра `ecx` внутри цикла `loop` может привести к некорректной работе программы. Я изменила текст программы, добавив модификацию регистра `ecx` в цикле.

Полученная программа: - Запускает бесконечный цикл при нечетном значении `N`. - Выводит только нечетные числа, если `N` четное.



```
mc [vnziborova@fedora]:~/work/arch-pc/lab08
lab8-1.asm [----] 26 L: [ 1+19 20/ 30] *(416 / 586b) 0010 0x
#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
; переход на `label`
call quit
```

Рис. 3.3: Программа в файле lab8-1.asm

```
4294933770
4294933768
4294933766
4294933764
42949337^C
vnziborova@fedora:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 6
5
3
1
vnziborova@fedora:~/work/arch-pc/lab08$
```

Рис. 3.4: Запуск программы lab8-1.asm

Для корректного использования регистра `ecx` в цикле, я добавила команды `push` и `pop`, чтобы временно сохранять значение регистра в стеке. Это позволило сохранить корректность работы программы. Внесла изменения в текст программы, создала исполняемый файл и проверила его работу.

Теперь программа: - Выводит числа от $N-1$ до 0. - Число проходов цикла соответствует значению N .

```
mc [vnziborova@fedora]:~/work/arch-pc/lab08
lab8-1.asm [----] 11 L:[ 1+24 25/ 31] *(559 / 675b) 0010 0x00A
#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 ; 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
call quit
```

Рис. 3.5: Программа в файле lab8-1.asm

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

Рис. 3.6: Запуск программы lab8-1.asm

3.2 Обработка аргументов командной строки

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

Создала исполняемый файл и запустила его, указав аргументы. Программа обработала 5 аргументов, разделенных пробелами (слова или числа).

```

lab8-2.asm      [----]  9 L: [  1+19  20/ 20] *(943 / 943b) <EOF>
#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

```

Рис. 3.7: Программа в файле lab8-2.asm

```

vnziborova@fedora:~/work/arch-pc/lab08$
vnziborova@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-2.asm
vnziborova@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 lab8-2.o -o lab8-2
vnziborova@fedora:~/work/arch-pc/lab08$ ./lab8-2 V e r o n i k a
V
e
r
o
n
i
k
a
vnziborova@fedora:~/work/arch-pc/lab08$

```

Рис. 3.8: Запуск программы lab8-2.asm

Рассмотрела пример программы, которая выводит сумму чисел, переданных в

качестве аргументов командной строки.

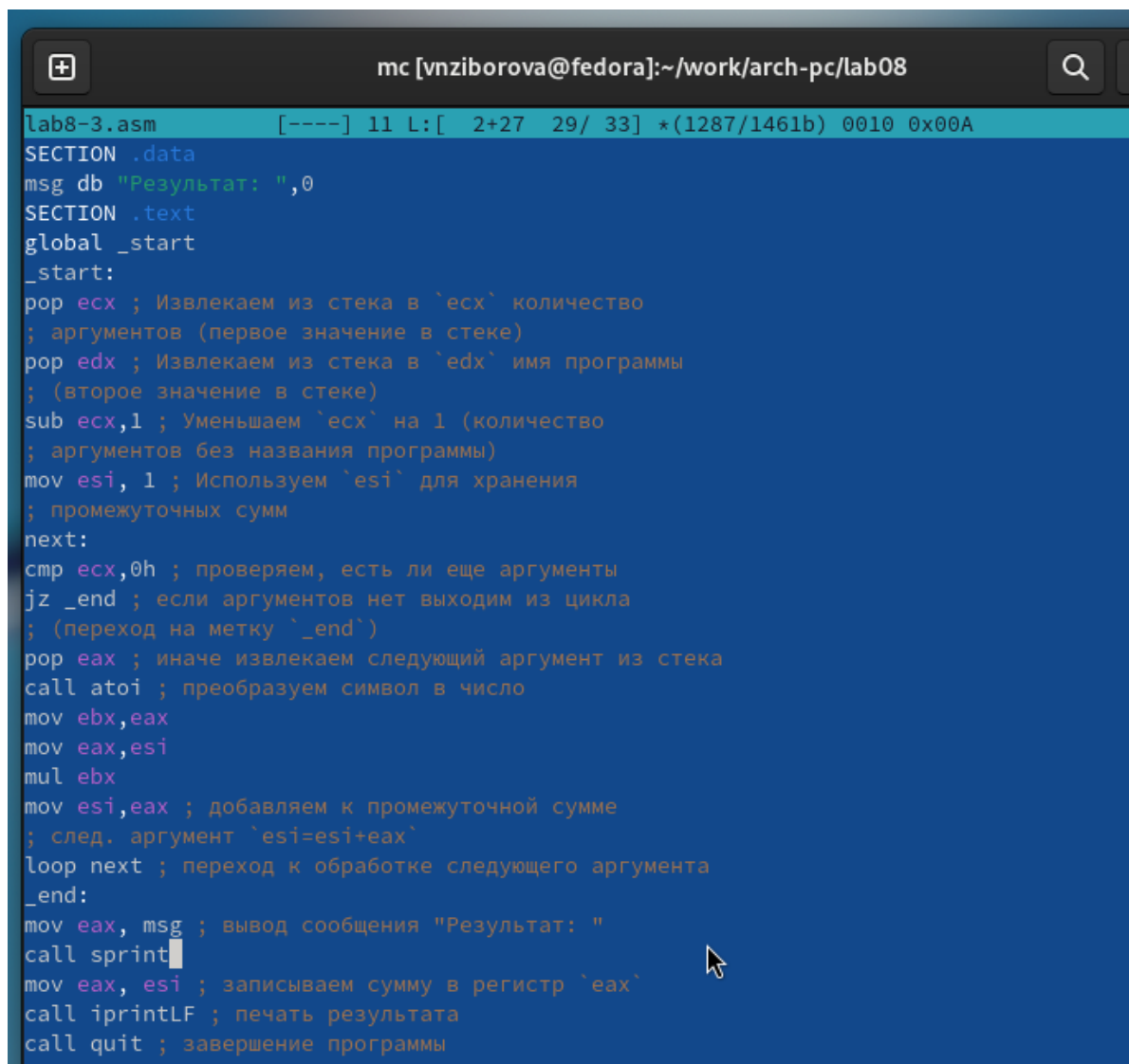
```
lab8-3.asm      [----] 44 L:[ 1+24 25/ 29] *(1243/1428b) 0010 0x00A
#include 'in_out.asm'
SECTION .data
msg db "Результат: ",0
SECTION .text
global _start
_start:
pop ecx ; Извлекаем из стека в `ecx` количество
; аргументов (первое значение в стеке)
pop edx ; Извлекаем из стека в `edx` имя программы
; (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. 3.9: Программа в файле lab8-3.asm

```
vnziborova@fedora:~/work/arch-pc/lab08$
vnziborova@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
vnziborova@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 lab8-3.o -o lab8-3
vnziborova@fedora:~/work/arch-pc/lab08$ ./lab8-3 3 4 5
Результат: 12
vnziborova@fedora:~/work/arch-pc/lab08$
```

Рис. 3.10: Запуск программы lab8-3.asm

Изменила текст программы из листинга 8.3, чтобы вычислять произведение аргументов.



```
lab8-3.asm [----] 11 L: [ 2+27 29/ 33] *(1287/1461b) 0010 0x00A
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
mov ebx,eax
mov eax,esi
mul ebx
mov 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. 3.11: Программа в файле lab8-3.asm


```

vnziborova@fedora:~/work/arch-pc/lab08$
vnziborova@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
vnziborova@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 lab8-3.o -o lab8-3
vnziborova@fedora:~/work/arch-pc/lab08$ ./lab8-3 3 4 5
Результат: 12
vnziborova@fedora:~/work/arch-pc/lab08$
vnziborova@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
vnziborova@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 lab8-3.o -o lab8-3
vnziborova@fedora:~/work/arch-pc/lab08$ ./lab8-3 3 4 5
Результат: 60
vnziborova@fedora:~/work/arch-pc/lab08$ █

```

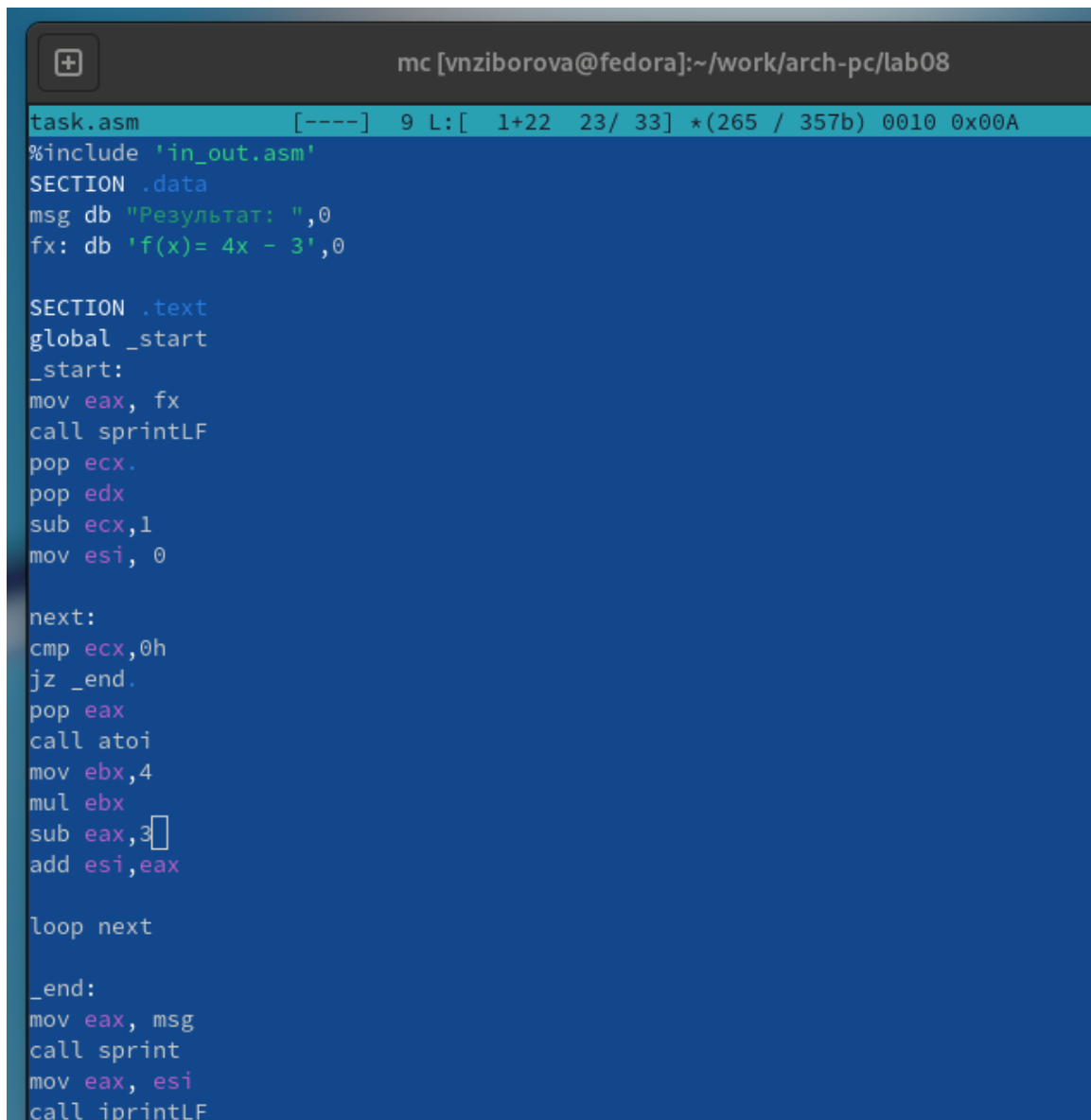
Рис. 3.12: Запуск программы lab8-3.asm

3.3 Задание для самостоятельной работы

Написала программу для нахождения суммы значений функции $f(x)$ для $x = x_1, x_2, \dots, x_n$. Программа должна выводить значение $f(x_1) + f(x_2) + \dots + f(x_n)$.

Значения x передаются в программу как аргументы. Для функции $f(x)$ выбрала вариант 6: $f(x) = 4x - 3$.

Создала исполняемый файл и проверила его работу на нескольких наборах значений x .



```
task.asm [----] 9 L: [ 1+22 23/ 33] *(265 / 357b) 0010 0x00A
#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 sprintLF
pop ecx
pop edx
sub ecx,1
mov esi, 0

next:
cmp ecx,0h
jz _end
pop eax
call atoi
mov ebx,4
mul ebx
sub eax,3
add esi,eax

loop next

_end:
mov eax, msg
call sprint
mov eax, esi
call iprintLF
```

Рис. 3.13: Программа в файле task.asm

Для проверки запустила программу сначала с одним аргументом: - При $x = 1$, $f(1) = 1$. - При $x = 2$, $f(2) = 5$.
Затем передала несколько аргументов и получила сумму значений функции.

```
vnziborova@fedora:~/work/arch-pc/lab08$  
vnziborova@fedora:~/work/arch-pc/lab08$ nasm -f elf task.asm  
vnziborova@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 task.o -o task  
vnziborova@fedora:~/work/arch-pc/lab08$ ./task  
f(x)= 4x - 3  
Результат: 0  
vnziborova@fedora:~/work/arch-pc/lab08$ ./task 1  
f(x)= 4x - 3  
Результат: 1  
vnziborova@fedora:~/work/arch-pc/lab08$ ./task 5  
f(x)= 4x - 3  
Результат: 17  
vnziborova@fedora:~/work/arch-pc/lab08$ ./task 2  
f(x)= 4x - 3  
Результат: 5  
vnziborova@fedora:~/work/arch-pc/lab08$ ./task 2 3 4 5  
f(x)= 4x - 3  
Результат: 44  
vnziborova@fedora:~/work/arch-pc/lab08$
```

Рис. 3.14: Запуск программы task.asm

4 Выводы

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

5 Ответы на вопросы

1. Опишите работу команды `loop`.

- Команда `loop` используется для организации циклов в NASM. Она уменьшает значение регистра `ECX` (или `CX` для 16-битных программ) на 1 и проверяет, не стало ли оно равным нулю. Если значение регистра не равно нулю, осуществляется переход по указанной метке. Если равно — выполнение программы продолжается с текущей позиции.

2. Как организовать цикл с помощью команд условных переходов, не прибегая к специальным командам управления циклами?

- Можно использовать прямую арифметическую модификацию регистра и условные переходы. Например:
 1. Инициализировать регистр-счетчик (например, `mov ecx, 10`).
 2. Внутри цикла уменьшать значение регистра (`dec ecx`).
 3. Проверять с помощью условной команды (`jnz, jz`) и организовывать переход на начало цикла.

3. Дайте определение понятия «стек».

- **Стек** — это структура данных, организованная по принципу «последним пришел — первым вышел» (LIFO, **Last In, First Out**). В контексте NASM стек используется для временного хранения данных, параметров функций, адресов возврата и других данных. Доступ к стеку осуществляется через команды `push` (помещение данных в стек) и `pop` (извлечение данных из стека).

4. Как осуществляется порядок выборки содержащихся в стеке данных?

- Данные извлекаются из стека в обратном порядке их помещения. Это соответствует принципу LIFO:
 1. Последний добавленный элемент извлекается первым.
 2. При каждом извлечении указатель стека (регистр ESP для 32-битных программ или SP для 16-битных) увеличивается.
 3. Для доступа к данным используются команды pop (снять элемент) или прямой доступ через указатель стека.