

# **Отчёта по лабораторной работе №9**

Уткина Алина Дмитриевна

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>4</b>
<b>2</b>	<b>Выполнение лабораторной работы</b>	<b>5</b>
2.1	Реализация циклов в NASM . . . . .	5
2.2	Обработка аргументов командной строки . . . . .	8
2.3	Самостоятельная работа . . . . .	10
<b>3</b>	<b>Выводы</b>	<b>12</b>

## Список иллюстраций

2.1	Программа вывода значений регистра ЕСХ . . . . .	6
2.2	Вывод значений регистра ЕСХ . . . . .	6
2.3	Изменение текста программы . . . . .	6
2.4	Результат работы программы (1) . . . . .	7
2.5	Результат работы программы (2) . . . . .	7
2.6	Добавление команд push и pop в программу . . . . .	7
2.7	Результат работы программы с командами push и pop . . . . .	8
2.8	Программа вывода аргументов командной строки . . . . .	9
2.9	Результат работы программы с аргументами . . . . .	9
2.10	Программа вычисления суммы аргументов командной строки . .	10
2.11	Результат работы программы вычисления суммы аргументов . .	10
2.12	Программа вычисления суммы значений функции от аргументов	11
2.13	Результат работы программы . . . . .	11

# 1 Цель работы

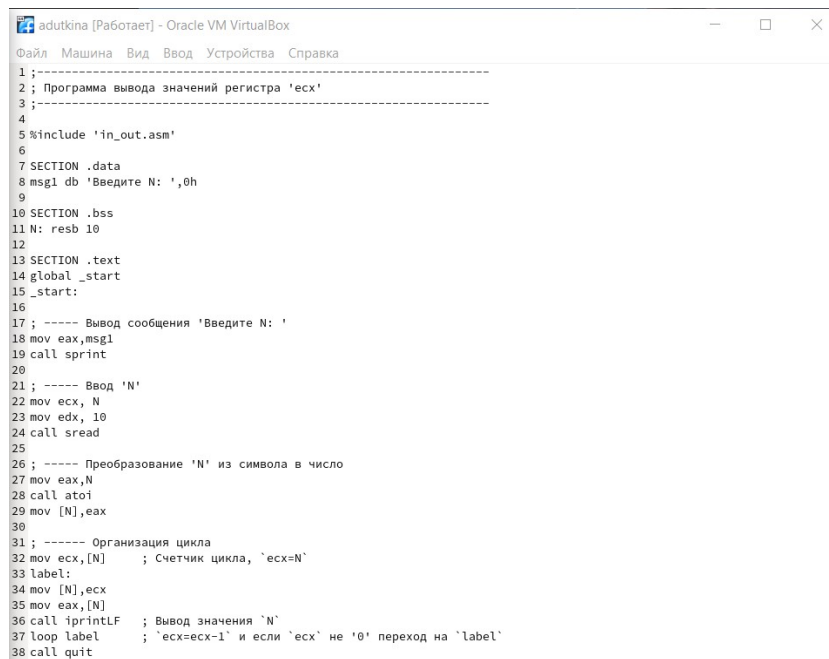
Целью данной работы является приобретение навыков написания программ с использованием циклов и обработкой аргументов командной строки.

## 2 Выполнение лабораторной работы

### 2.1 Реализация циклов в NASM

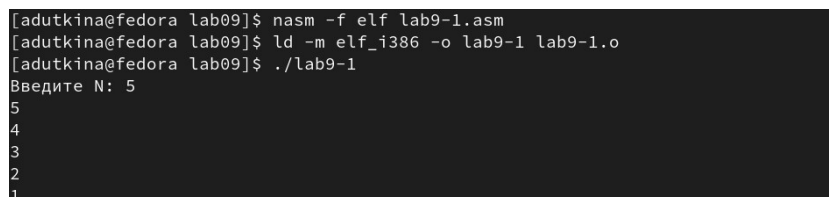
Создадим каталог для программ лабораторной работы № 9, перейдем в него и создадим файл lab9-1.asm.

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 изучим текст программы из листинга 9.1 и введем его в файл lab9-1.asm (рис. 2.1). Создадим исполняемый файл и проверим его работу (рис. 2.2).



```
1 ;-----
2 ; Программа вывода значений регистра 'ecx'
3 ;-----
4
5 %include 'in_out.asm'
6
7 SECTION .data
8 msg1 db 'Введите N: ',0h
9
10 SECTION .bss
11 N: resb 10
12
13 SECTION .text
14 global _start
15 _start:
16
17 ; ----- Вывод сообщения 'Введите N: '
18 mov eax,msg1
19 call sprint
20
21 ; ----- Ввод 'N'
22 mov ecx, N
23 mov edx, 10
24 call sread
25
26 ; ----- Преобразование 'N' из символа в число
27 mov eax,N
28 call atoi
29 mov [N],eax
30
31 ; ----- Организация цикла
32 mov ecx,[N] ; Счетчик цикла, 'ecx=N'
33 label:
34 mov [N],ecx
35 mov eax,[N]
36 call iprintLF ; Вывод значения 'N'
37 loop label ; 'ecx=ecx-1' и если 'ecx' не '0' переход на 'label'
38 call quit
```

Рис. 2.1: Программа вывода значений регистра ECX



```
[adutkina@fedora lab09]$ nasm -f elf lab9-1.asm
[adutkina@fedora lab09]$ ld -m elf_i386 -o lab9-1 lab9-1.o
[adutkina@fedora lab09]$ ./lab9-1
Введите N: 5
5
4
3
2
1
```

Рис. 2.2: Вывод значений регистра ECX

Данный пример показывает, что использование регистра `ecx` в теле цикла `loop` может привести к некорректной работе программы. Изменим текст программы, добавив изменение значения регистра `ecx` в цикле (рис. 2.3).

```
34 label:
35 sub ecx, 1 ; 'ecx=ecx-1'
36 mov [N],ecx
37 mov eax,[N]
38 call iprintLF ; Вывод значения 'N'
39
40 loop label
41 call quit
```

Рис. 2.3: Изменение текста программы

Создадим исполняемый файл и проверим его работу (рис. 2.4). Регистр `ecx` принимает значения на 2 меньше предыдущих. Также, из-за того, что мы ввели

нечетное число, 0 не попадает в проверку условия, то есть при одной проверке у нас значение регистра равно 1, а в следующей - 1, значит происходит заикливание. Если мы введем четное число, цикл остановится на 0, сделав в два раза меньше проходов, чем нужно (рис. 2.5).

```
[adutkina@fedora lab09]$ ./lab9-1
Введите N: 5
4
2
0
4294967294
4294967292
4294967290
4294967288
4294967286
```

Рис. 2.4: Результат работы программы (1)

```
[adutkina@fedora lab09]$ ./lab9-1
Введите N: 4
3
1
[adutkina@fedora lab09]$
```

Рис. 2.5: Результат работы программы (2)

Для использования регистра `ecx` 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` (рис. 2.6).

```
34 label:
35 push ecx      ; добавление значения ecx в стек
36 sub ecx, 1    ; 'ecx=ecx-1'
37 mov [N],ecx
38 mov eax,[N]
39 call iprintLF ; Вывод значения 'N'
40 pop ecx      ; извлечение ecx из стека
41
42 loop label
```

Рис. 2.6: Добавление команд `push` и `pop` в программу

Создадим исполняемый файл и проверим его работу (рис. 2.7). Количество проходов цикла соответствует введенному значению.

```
[adutkina@fedora lab09]$ ./lab9-1
Введите N: 5
4
3
2
1
0
[adutkina@fedora lab09]$
```

Рис. 2.7: Результат работы программы с командами push и pop

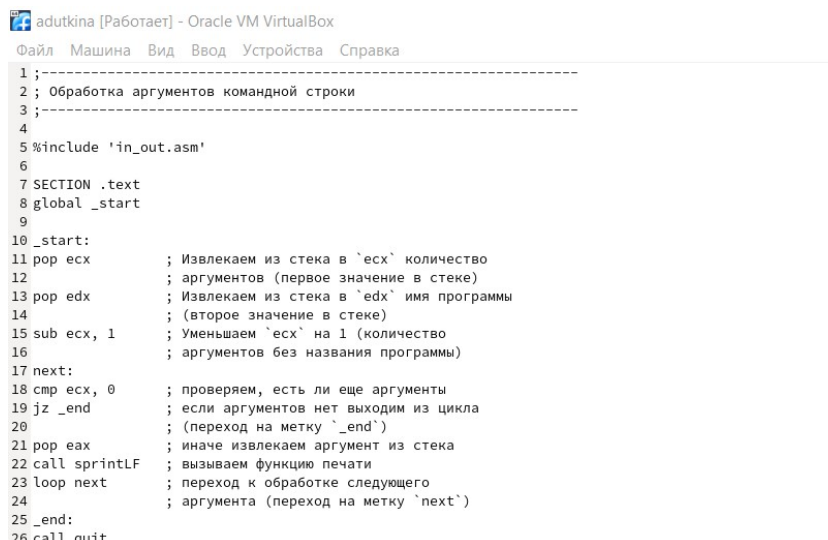
## 2.2 Обработка аргументов командной строки

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.

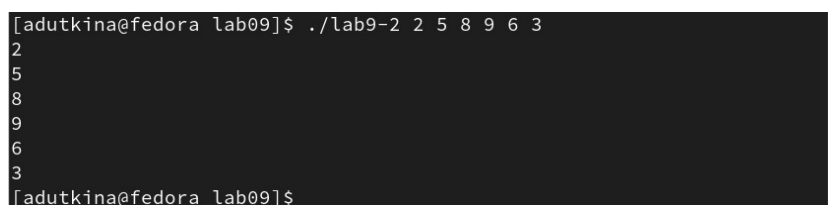
Таким образом, для того чтобы использовать аргументы в программе, их просто нужно извлечь из стека. Обработку аргументов нужно проводить в цикле. Т.е. сначала нужно извлечь из стека количество аргументов, а затем циклично для каждого аргумента выполнить логику программы. В качестве примера рассмотрим программу, которая выводит на экран аргументы командной строки. Внимательно изучим текст программы из листинга 9.2 и введем его в файл lab9-2.asm (рис. 2.8). Создадим исполняемый файл и запустим его, указав аргументы (рис. 2.9). Все аргументы были обработаны программой.





```
adutkina [Работает] - Oracle VM VirtualBox
Файл  Машина  Вид  Ввод  Устройства  Справка
1;
2; Обработка аргументов командной строки
3;
4
5%include 'in_out.asm'
6
7SECTION .text
8global _start
9
10_start:
11 pop ecx          ; Извлекаем из стека в `ecx` количество
12                 ; аргументов (первое значение в стеке)
13 pop edx          ; Извлекаем из стека в `edx` имя программы
14                 ; (второе значение в стеке)
15 sub ecx, 1       ; Уменьшаем `ecx` на 1 (количество
16                 ; аргументов без названия программы)
17 next:
18 cmp ecx, 0       ; проверяем, есть ли еще аргументы
19 jz _end          ; если аргументов нет выходим из цикла
20                 ; (переход на метку `_end`)
21 pop eax          ; иначе извлекаем аргумент из стека
22 call sprintf     ; вызываем функцию печати
23 loop next        ; переход к обработке следующего
24                 ; аргумента (переход на метку `next`)
25 _end:
26 call quit
```

Рис. 2.8: Программа вывода аргументов командной строки



```
[adutkina@fedora lab09]$ ./lab9-2 2 5 8 9 6 3
2
5
8
9
6
3
[adutkina@fedora lab09]$
```

Рис. 2.9: Результат работы программы с аргументами

Рассмотрим еще один пример программы которая выводит сумму чисел, кото-  
рые передаются в программу как аргументы. Создадим файл lab9-3.asm и введем  
в него текст программы из листинга 9.3 (рис. 2.10). Создадим исполняемый файл  
и запустим его, указав аргументы (рис. 2.11).

Рис. 2.10: Программа вычисления суммы аргументов командной строки

```
[adutkina@fedora lab09]$ ./lab9-3 12 13 7 10 5
Результат: 47
[adutkina@fedora lab09]$ ./lab9-3 10 15 20 25
Результат: 70
[adutkina@fedora lab09]$
```

Рис. 2.11: Результат работы программы вычисления суммы аргументов

## 2.3 Самостоятельная работа

Напишем программу, которая находит сумму значений функции  $f(x) = 6x + 13$  для  $x = x_1, x_2, \dots, x_n$ , т.е. программа должна выводить значение  $f(x_1) + f(x_2) + \dots + f(x_n)$ , где значения  $x_i$  передаются как аргументы (рис. 2.12). Создадим исполняемый файл и проверим его работу на нескольких наборах  $x = x_1, x_2, \dots, x_n$  (рис. 2.13). Программа работает при различном количестве аргументов верно.

```
adutkina [Работает] - Oracle VM VirtualBox
Файл  Машина  Вид  Ввод  Устройства  Справка

1 ;-----
2 ; Вычисления суммы значений функции от различных аргументов
3 ;-----
4
5 %include 'in_out.asm'
6
7 SECTION .data
8 msg1 db "Функция: f(x) = 6x + 13", 0
9 msg db "Результат: ",0
10
11 SECTION .text
12 global _start
13
14 _start:
15
16 mov eax, msg1 ; вывод сообщения "Функция: ..."
17 call sprintf
18
19 pop ecx ; Извлекаем из стека в 'ecx' количество
20 ; аргументов (первое значение в стеке)
21 pop edx ; Извлекаем из стека в 'edx' имя программы
22 ; (второе значение в стеке)
23
24 sub ecx,1 ; Уменьшаем 'ecx' на 1 (количество
25 ; аргументов без названия программы)
26 mov esi, 0 ; Используем 'esi' для хранения
27 ; промежуточных сумм
28 next:
29 cmp ecx,0h ; проверяем, есть ли еще аргументы
30 jz _end ; если аргументов нет выходим из цикла
31 ; (переход на метку '_end')
32
33 pop eax ; иначе извлекаем следующий аргумент из стека
34 call atoi ; преобразуем символ в число
35
36 mov ebx, 6
37 mul ebx ; EAX=EAX*6
38 add eax, 13 ; EAX=EAX+13
39
40 add esi,eax ; добавляем к промежуточной сумме
41 ; след. аргумент 'esi=esi+eax'
42 loop next ; переход к обработке следующего аргумента
43
44 _end:
45 mov eax, msg ; вывод сообщения "Результат: "
46 call sprintf
47 mov eax, esi ; записываем сумму в регистр 'eax'
48 call iprintf ; печать результата
49 call quit ; завершение программы
50
```

Рис. 2.12: Программа вычисления суммы значений функции от аргументов

```
[adutkina@fedora lab09]$ ./var9 1
Функция: f(x) = 6x + 13
Результат: 19
[adutkina@fedora lab09]$ ./var9 1 2
Функция: f(x) = 6x + 13
Результат: 44
[adutkina@fedora lab09]$ ./var9 1 2 3
Функция: f(x) = 6x + 13
Результат: 75
[adutkina@fedora lab09]$
```

Рис. 2.13: Результат работы программы

## **3 Выводы**

В ходе данной работы были приобретены навыки написания программ с использованием циклов и обработкой аргументов командной строки.