

# **Лабораторная работа №8**

**Программирование цикла. Обработка аргументов командной строки.**

**Перфилов Александр Константинович | группа: НПИбд 02-23**

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Выполнение лабораторной работы</b>	<b>6</b>
<b>3</b>	<b>Самостоятельная работа</b>	<b>17</b>
<b>4</b>	<b>Выводы</b>	<b>20</b>

## Список иллюстраций

2.1	Рис 2.1.1: Создание каталога и файла .asm . . . . .	6
2.2	Рис 2.1.2: Программа . . . . .	7
2.3	Рис 2.1.3: Создание исполняемого файла и проверка работы . . .	7
2.4	Рис 2.1.4: Измененная программа . . . . .	8
2.5	Рис 2.1.5: Создание исполняемого файла и проверка работы . . .	9
2.6	Рис 2.1.6: Измененная программа . . . . .	10
2.7	Рис 2.1.7: Создание исполняемого файла и проверка работы . . .	11
2.8	Рис 2.2.1: Создание файла .asm . . . . .	11
2.9	Рис 2.2.2: Программа . . . . .	12
2.10	Рис 2.2.3: Создание исполняемого файла и проверка работы с указанием аргументов . . . . .	12
2.11	Рис 2.2.4: Создание файла .asm . . . . .	13
2.12	Рис 2.2.5: Программа . . . . .	13
2.13	Рис 2.2.6: Создание исполняемого файла и проверка работы с указанием аргументов . . . . .	14
2.14	Рис 2.2.7: Измененная программа . . . . .	15
2.15	Рис 2.2.8: Создание исполняемого файла и проверка работы с указанием аргументов . . . . .	16
3.1	Рис 3.1.1: Создание файла . . . . .	17
3.2	Рис 3.1.2: Программа для задания . . . . .	18
3.3	Рис 3.1.3: Проверка программы . . . . .	19
3.4	Рис 3.1.4: Загрузка файлов на github . . . . .	19

## **Список таблиц**

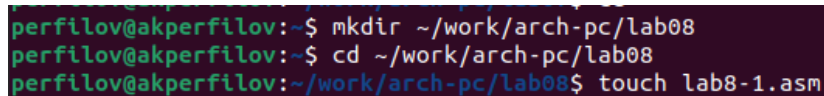
# 1 Цель работы

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

## 2 Выполнение лабораторной работы

### Реализация циклов в NASM

Создадим каталог для программ лабораторной работы № 8, перейдем в него и создадим файл lab8-1.asm



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

Рис. 2.1: Рис 2.1.1: Создание каталога и файла .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 (Листинг 8.1).

Введем в файл `lab8-1.asm` текст программы из листинга 8.1.

```

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

```

Рис. 2.2: Рис 2.1.2: Программа

Создадим исполняемый файл и запустим его. Результат работы данной программы будет следующим:

```

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

```

Рис. 2.3: Рис 2.1.3: Создание исполняемого файла и проверка работы

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:

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

Рис. 2.4: Рис 2.1.4: Измененная программа

Создадим исполняемый файл и проверим его работу.



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

Рис. 2.5: Рис 2.1.5: Создание исполняемого файла и проверка работы

Если ввести 4, то число проходов цикла не будет соответствовать введенному с клавиатуры значению

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`:

```

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

```

Рис. 2.6: Рис 2.1.6: Измененная программа

Создадим исполняемый файл и проверим его работу.

```

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

```

Рис. 2.7: Рис 2.1.7: Создание исполняемого файла и проверка работы

Число проходов цикла будет соответствовать введенному значению с клавиатуры

### 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 (Листинг 8.2)

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

```

perfilov@akperfilov:~/work/arch-pc/lab08$ touch lab8-2.asm

```

Рис. 2.8: Рис 2.2.1: Создание файла .asm

```

1 %include 'in_out.asm'
2 SECTION .text
3 global _start
4 _start:
5 pop ecx ; Извлекаем из стека в `ecx` количество
6 ; аргументов (первое значение в стеке)
7 pop edx ; Извлекаем из стека в `edx` имя программы
8 ; (второе значение в стеке)
9 sub ecx, 1 ; Уменьшаем `ecx` на 1 (количество
10 ; аргументов без названия программы)
11 next:
12 cmp ecx, 0 ; проверяем, есть ли еще аргументы
13 jz _end ; если аргументов нет выходим из цикла
14 ; (переход на метку `_end`)
15 pop eax ; иначе извлекаем аргумент из стека
16 call sprintf ; вызываем функцию печати
17 loop next ; переход к обработке следующего
18 ; аргумента (переход на метку `next`)
19 _end:
20 call quit

```

Рис. 2.9: Рис 2.2.2: Программа

Создадим исполняемый файл и запустим его, указав аргументы:

```

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

```

Рис. 2.10: Рис 2.2.3: Создание исполняемого файла и проверка работы с указанием аргументов

Всего было обработано 4 аргумента, так как второй и третий аргумент не были взяты в кавычки, в отличие от 4-го аргумента

Рассмотрим еще один пример программы которая выводит сумму чисел, которые передаются в программу как аргументы. Создадим файл lab8-3.asm в каталоге ~/work/arch-pc/lab08 и введем в него текст программы из листинга 8.3.

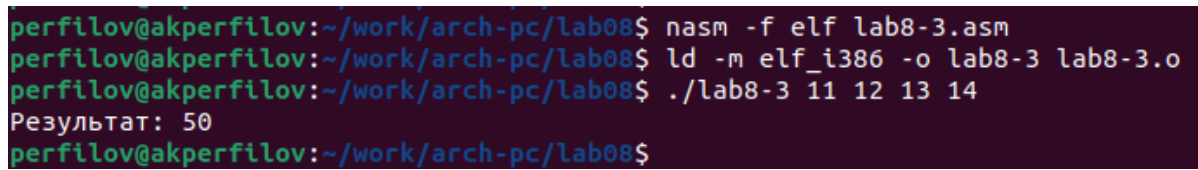
```
perfilov@akperfilov:~/work/arch-pc/lab08$ touch lab8-3.asm
```

Рис. 2.11: Рис 2.2.4: Создание файла .asm

```
1 %include 'in_out.asm'
2 SECTION .data
3 msg db "Результат: ",0
4 SECTION .text
5 global _start
6 _start:
7 pop ecx ; Извлекаем из стека в `ecx` количество
8 ; аргументов (первое значение в стеке)
9 pop edx ; Извлекаем из стека в `edx` имя программы
10 ; (второе значение в стеке)
11 sub ecx,1 ; Уменьшаем `ecx` на 1 (количество
12 ; аргументов без названия программы)
13 mov esi, 0 ; Используем `esi` для хранения
14 ; промежуточных сумм
15 next:
16 cmp ecx,0h ; проверяем, есть ли еще аргументы
17 jz _end ; если аргументов нет выходим из цикла
18 ; (переход на метку `_end`)
19 pop eax ; иначе извлекаем следующий аргумент из стека
20 call atoi ; преобразуем символ в число
21 add esi,eax ; добавляем к промежуточной сумме
22 ; след. аргумент `esi=esi+eax`
23 loop next ; переход к обработке следующего аргумента
24 _end:
25 mov eax, msg ; вывод сообщения "Результат: "
26 call sprint
27 mov eax, esi ; записываем сумму в регистр `eax`
28 call iprintLF ; печать результата
29 call quit ; завершение программы
```

Рис. 2.12: Рис 2.2.5: Программа

Создадим исполняемый файл и запустим его, указав аргументы.



```
perfilov@akperfilov:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
perfilov@akperfilov:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
perfilov@akperfilov:~/work/arch-pc/lab08$ ./lab8-3 11 12 13 14
Результат: 50
perfilov@akperfilov:~/work/arch-pc/lab08$
```

Рис. 2.13: Рис 2.2.6: Создание исполняемого файла и проверка работы с указанием аргументов

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

```

1 %include 'in_out.asm'
2 SECTION .data
3 msg db "Результат: ",0
4 SECTION .text
5 global _start
6 _start:
7 pop ecx ; Извлекаем из стека в `ecx` количество
8 ; аргументов (первое значение в стеке)
9 pop edx ; Извлекаем из стека в `edx` имя программы
10 ; (второе значение в стеке)
11 sub ecx,1 ; Уменьшаем `ecx` на 1 (количество
12 ; аргументов без названия программы)
13 mov esi, 1 ; Используем `esi` для хранения
14 ; промежуточных сумм
15 next:
16 cmp ecx,0h ; проверяем, есть ли еще аргументы
17 jz _end ; если аргументов нет выходим из цикла
18 ; (переход на метку `_end`)
19 pop eax ; иначе извлекаем следующий аргумент из стека
20 call atoi ; преобразуем символ в число
21 mul esi
22 mov esi,eax ; добавляем к промежуточной сумме
23 ; след. аргумент `esi=esi+eax`
24 loop next ; переход к обработке следующего аргумента
25 _end:
26 mov eax, msg ; вывод сообщения "Результат: "
27 call sprint
28 mov eax, esi ; записываем сумму в регистр `eax`
29 call iprintLF ; печать результата
30 call quit ; завершение программы

```

Рис. 2.14: Рис 2.2.7: Измененная программа

```
perfilov@akperfilov:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
perfilov@akperfilov:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
perfilov@akperfilov:~/work/arch-pc/lab08$ ./lab8-3 11 12 13 14
Результат: 24024
perfilov@akperfilov:~/work/arch-pc/lab08$
```

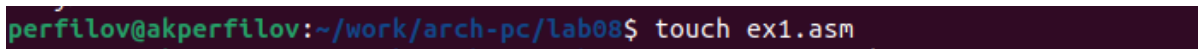
Рис. 2.15: Рис 2.2.8: Создание исполняемого файла и проверка работы с указанием аргументов



### 3 Самостоятельная работа

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

Создадим новый файл `ex1.asm` и напомним программу нахождения суммы значения функции  $f(x)$  для варианта 19 ()).



```
perfilov@akperfilov:~/work/arch-pc/lab08$ touch ex1.asm
```

Рис. 3.1: Рис 3.1.1: Создание файла

Возьмем за основу код из `lab8-3.asm` и переделаем его под 1-ое задание

```

1 %include 'in_out.asm'
2
3 SECTION .data
4 msg db "Результат: ",0
5
6 SECTION .text
7 global _start
8 _start:
9
10 pop ecx
11
12 pop edx
13
14 sub ecx,1
15
16 mov esi, 0
17
18 next:
19 cmp ecx,0h
20 jz _end
21
22 pop eax
23 call atoi
24 mov ebx,8
25 mul ebx
26 sub eax,3
27 add esi,eax
28 loop next
29 _end:
30
31 mov eax, msg
32 call sprint
33 mov eax, esi
34 call iprintLF
35 call quit

```

Рис. 3.2: Рис 3.1.2: Программа для задания

Проверим программу с несколькими значениями x

```
perfilov@akperfilov:~/work/arch-pc/lab08$ nasm -f elf ex1.asm
perfilov@akperfilov:~/work/arch-pc/lab08$ ld -m elf_i386 -o ex1 ex1.o
perfilov@akperfilov:~/work/arch-pc/lab08$ ./ex1 1 2 3 4
Результат: 68
perfilov@akperfilov:~/work/arch-pc/lab08$ ./ex1 2 2 2 2
Результат: 52
perfilov@akperfilov:~/work/arch-pc/lab08$ ./ex1 12 41 -2 1
Результат: 420
perfilov@akperfilov:~/work/arch-pc/lab08$
```

Рис. 3.3: Рис 3.1.3: Проверка программы

Как видно по (Рис 3.1.3) программа работает верно

Загрузим все файлы на github

```
perfilov@akperfilov:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs$ git add .
perfilov@akperfilov:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs$ git commit -am 'feat(main): add files lab-8'
[master f7a534c] feat(main): add files lab-8
 4 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 labs/lab07/report/image.zip
 rename "labs/lab08/report/image/\320\240\320\270\321\201 3.1.2 .png" => "labs/lab08/report/image/\320\240\320\270\321\201 3.1.2.png" (100%)
 create mode 100644 labs/lab08/report/report.docx
 create mode 100644 labs/lab08/report/report.pdf
perfilov@akperfilov:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs$ git push
Перечисление объектов: 18, готово.
Подсчет объектов: 100% (18/18), готово.
При сжатии изменений используется до 4 потоков
Сжатие объектов: 100% (11/11), готово.
Запись объектов: 100% (11/11), 2.36 Миб | 2.98 Миб/с, готово.
Всего 11 (изменений 5), повторно использовано 0 (изменений 0), повторно использовано пакетов 0
remote: Resolving deltas: 100% (5/5), completed with 5 local objects.
To github.com:AlexanderPerfilovKonstantinivich/study_2023-2024_arh-pc.git
 da5c773..f7a534c master -> master
perfilov@akperfilov:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs$
```

Рис. 3.4: Рис 3.1.4: Загрузка файлов на github

## 4 Выводы

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