

Отчёт по лабораторной работе №8

Дисциплина: архитектура компьютеров и операционные системы

Глущенко Евгений Игоревич

1 Цель работы

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

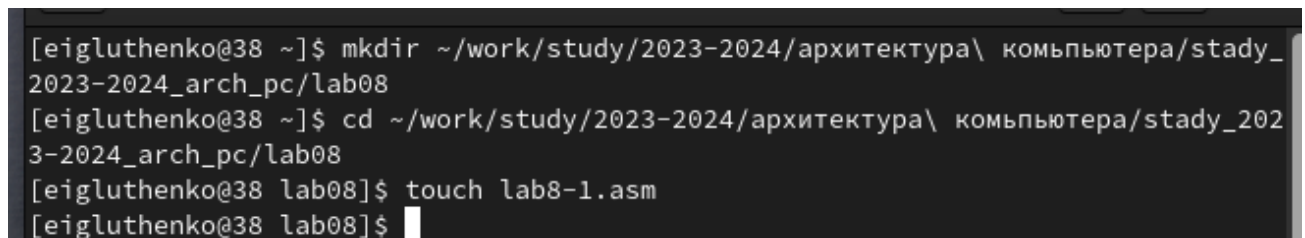
2 Задание

1. Реализация циклов в NASM.
2. Обработка аргументов командной строки.
3. Задание для самостоятельной работы.

3 Выполнение лабораторной работы

3.1 Реализация циклов в NASM


Создаю каталог для лабораторной работы № 8, перехожу в него и создаю файл lab8-1.asm. (рис. 15).



```
[eigluthenko@38 ~]$ mkdir ~/work/study/2023-2024/архитектура\ компьютера/stady_2023-2024_arch_pc/lab08
[eigluthenko@38 ~]$ cd ~/work/study/2023-2024/архитектура\ компьютера/stady_2023-2024_arch_pc/lab08
[eigluthenko@38 lab08]$ touch lab8-1.asm
[eigluthenko@38 lab08]$
```

Рисунок 1: Создание файлов для лабораторной работы

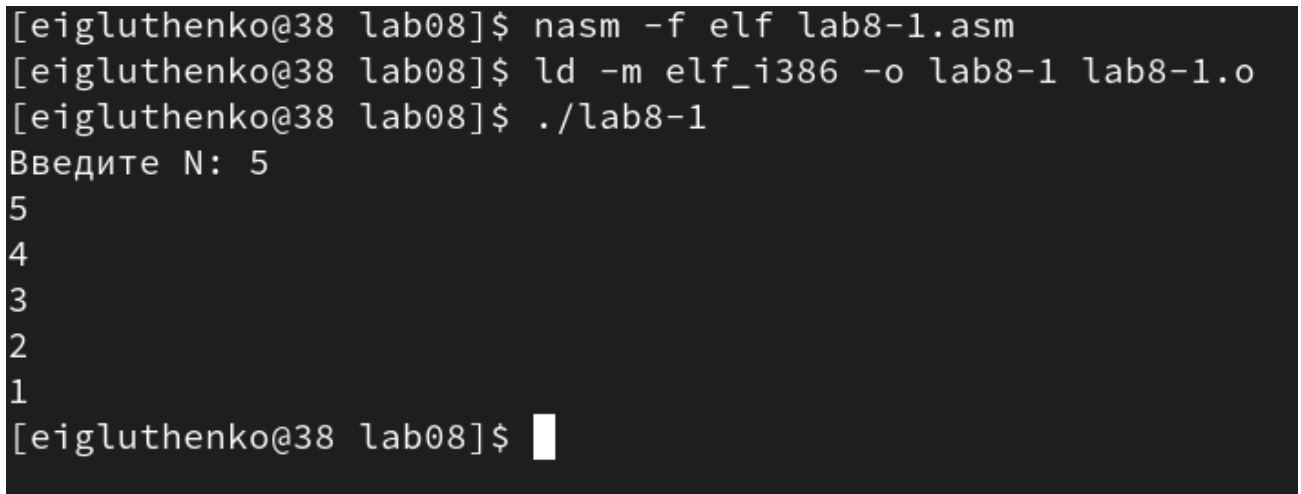
Ввожу в файл lab8-1.asm текст программы. (рис. 15).



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

Рисунок 2: Ввод текста из листинга 8.1

Создаю исполняемый файл и проверяю его работу. (рис. 15).



```
[eigluthenko@38 lab08]$ nasm -f elf lab8-1.asm
[eigluthenko@38 lab08]$ ld -m elf_i386 -o lab8-1 lab8-1.o
[eigluthenko@38 lab08]$ ./lab8-1
Введите N: 5
5
4
3
2
1
[eigluthenko@38 lab08]$
```

Рисунок 3: Запуск исполняемого файла

Данная программа выводит числа от N до 1 включительно.

Изменяю текст программы, добавив изменение значения регистра ecx в цикле. (рис. 15).

```

~/work/study/2023-2024/архитектура компьютера/stady_2023-2024_arch_p
1 %include 'in_out.asm'
2 SECTION .data
3 msg1 db 'Введите N: ',0h
4 SECTION .bss
5 N: resb 10
6 SECTION .text
7 global _start
8 _start:
9 ; ----- Вывод сообщения 'Введите N: '
10 mov eax,msg1
11 call sprint
12 ; ----- Ввод 'N'
13 mov ecx, N
14 mov edx, 10
15 call sread
16 ; ----- Преобразование 'N' из символа в число
17 mov eax,N
18 call atoi
19 mov [N],eax
20 ; ----- Организация цикла
21 mov ecx,[N] ; Счетчик цикла, `ecx=N`
22 label:
23 sub ecx,1 ; `ecx=ecx-1`
24 mov [N],ecx
25 mov eax,[N]
26 call iprintLF; Вывод значения `N`
27 loop label ; `ecx=ecx-1` и если `ecx` не '0'
28 ; переход на `label`
29 call quit

```

Рисунок 4: Изменение текста программы

Создаю исполняемый файл и проверяю его работу. (рис. 15).

```

675
673
671
669
667
665
663
661

```

Рисунок 5: Запуск обновленной программы

В данном случае число проходов цикла не соответствует введенному с клавиатуры значению.

Вношу изменения в текст программы, добавив команды push и pop для сохранения значения счетчика цикла loop. (рис. 15).

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

Рисунок 6: Изменение текста программы

Создаю исполняемый файл и проверяю его работу.(рис. 15).

```

[eigluthenko@38 lab08]$ nasm -f elf lab8-1.asm
[eigluthenko@38 lab08]$ ld -m elf_i386 -o lab8-1 lab8-1.o
[eigluthenko@38 lab08]$ ./lab8-1
Введите N: 5
4
3
2
1
0
[eigluthenko@38 lab08]$

```

Рисунок 7: Запуск исполняемого файла

В данном случае число проходов цикла соответствует введенному с клавиатуры значению и выводит числа от N-1 до 0 включительно.

3.2 Обработка аргументов командной строки

Создаю файл lab8-2.asm в каталоге и ввожу в него текст программы. (рис. 15).

```

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 sprintLF ; вызываем функцию печати
17 loop next ; переход к обработке следующего
18 ; аргумента (переход на метку `next`)
19 _end:
20 call quit

```

Рисунок 8: Ввод текста программы из листинга 8.2

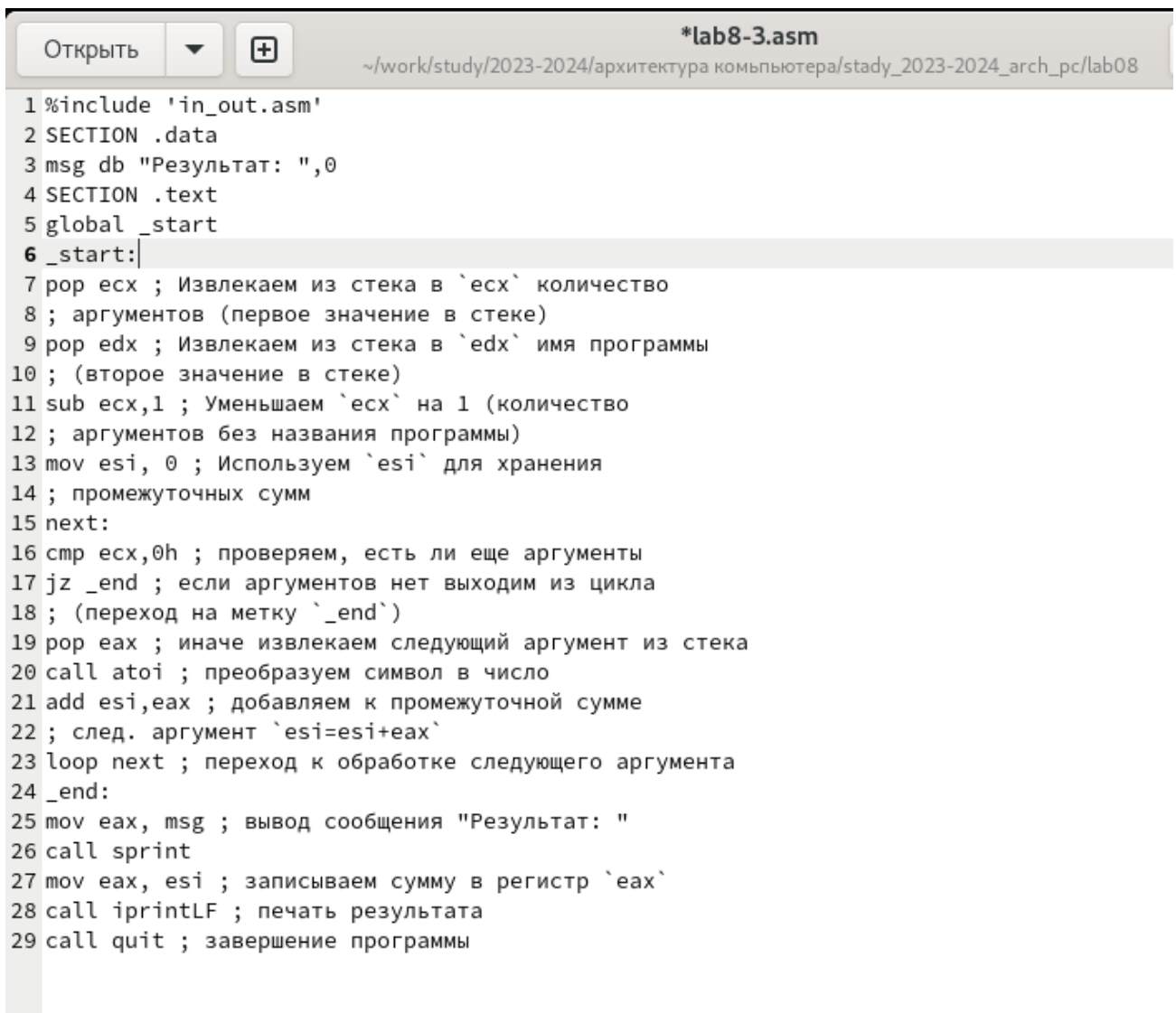
Создаю исполняемый файл и запускаю его, указав нужные аргументы. (рис. 15).

```
[eigluthenko@38 lab08]$ nasm -f elf lab8-2.asm
[eigluthenko@38 lab08]$ ld -m elf_i386 -o lab8-2 lab8-2.o
[eigluthenko@38 lab08]$ ./lab8-2 аргумент1 аргумент 2 'аргумент 3'
аргумент1
аргумент
2
аргумент 3
[eigluthenko@38 lab08]$
```

Рисунок 9: Запуск исполняемого файла

Программа вывела 4 аргумента, так как аргумент 2 не взят в кавычки, в отличии от аргумента 3, поэтому из-за пробела программа считывает “2” как отдельный аргумент.

Рассмотрим пример программы, которая выводит сумму чисел, которые передаются в программу как аргументы. Создаю файл lab8-3.asm в каталоге и ввожу в него текст программы. (рис. [15](#)).

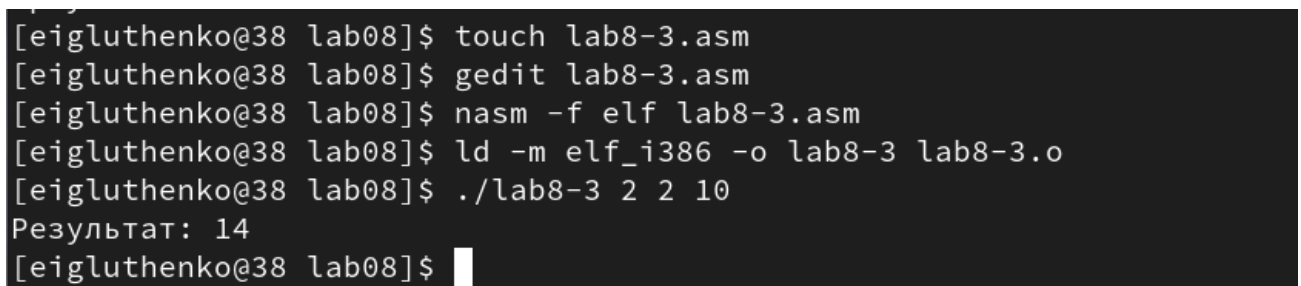


```
*lab8-3.asm
~/work/study/2023-2024/архитектура компьютера/stady_2023-2024_arch_pc/lab08

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
```

Рисунок 10: Ввод текста программы из листинга 8.3

Создаю исполняемый файл и запускаю его, указав аргументы. (рис. 15).



```
[eigluthenko@38 lab08]$ touch lab8-3.asm
[eigluthenko@38 lab08]$ gedit lab8-3.asm
[eigluthenko@38 lab08]$ nasm -f elf lab8-3.asm
[eigluthenko@38 lab08]$ ld -m elf_i386 -o lab8-3 lab8-3.o
[eigluthenko@38 lab08]$ ./lab8-3 2 2 10
Результат: 14
[eigluthenko@38 lab08]$
```

Рисунок 11: Запуск исполняемого файла

Изменяю текст программы из листинга 8.3 для вычисления произведения аргументов командной строки. (рис. 15).

```

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

```

Рисунок 12: Изменение текста программы

Создаю исполняемый файл и запускаю его, указав аргументы. (рис. 15).

```

[eigluthenko@38 lab08]$ gedit lab8-3.asm
[eigluthenko@38 lab08]$ nasm -f elf lab8-3.asm
[eigluthenko@38 lab08]$ ld -m elf_i386 -o lab8-3 lab8-3.o
[eigluthenko@38 lab08]$ ./lab8-3 2 2 10
Результат: 40
[eigluthenko@38 lab08]$

```

Рисунок 13: Запуск исполняемого файла

3.3 Задание для самостоятельной работы

Пишу текст программы, которая находит сумму значений функции $f(x) = 5 \cdot (2 + x)$ в соответствии с моим номером варианта (10). (рис. 15).


```

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 imul 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

```

Рисунок 14: Текст программы

Создаю исполняемый файл и проверьте его работу на нескольких наборах $x = x_1, x_2, \dots, x_n$. (рис. 15).

```

[eigluthenko@38 lab08]$ nasm -f elf task.asm
[eigluthenko@38 lab08]$ ld -m elf_i386 -o task task.o
[eigluthenko@38 lab08]$ ./task 1 2 3
Результат: 6
[eigluthenko@38 lab08]$ ./task 1 2 3 33
Результат: 198
[eigluthenko@38 lab08]$ ./task 1 2 3 4
Результат: 24
[eigluthenko@38 lab08]$

```

Рисунок 15: Запуск исполняемого файла и проверка его работы

Программа работает корректно.

Текст программы:

```
%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 eax,2
```

```
imul eax,3
```

```
add esi,eax ; добавляем к промежуточной сумме
```

```
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`

4 Выводы

Благодаря данной лабораторной работе я получил навыки написания программ с использованием циклов и обработкой аргументов командной строки.