

# **Лабораторная работа №8**

**Программирование цикла. Обработка аргументов командной строки**

Хасанов Тимур

# Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
3	Выполнение задания для самостоятельной работы	17
4	Выводы	20

## Список иллюстраций

2.1	Создание файла lab8-1.asm . . . . .	6
2.2	Запуск Midnight commander . . . . .	7
2.3	Вставка кода из файла листинга 8.1 . . . . .	8
2.4	Копирование файла in_out.asm в рабочую директорию . . . . .	9
2.5	Сборка программы из файла lab8-1.asm и её запуск . . . . .	9
2.6	Изменение файла lab8-1.asm . . . . .	10
2.7	Повторная сборка программы из файла lab8-1.asm и её запуск . .	10
2.8	Результат вывода . . . . .	11
2.9	Результат вывода для чётного N . . . . .	12
2.10	Редактирование файла lab8-1.asm . . . . .	12
2.11	Повторная сборка программы из файла lab8-1.asm и её запуск . .	13
2.12	Создание второго файла: lab8-2.asm . . . . .	13
2.13	Запись кода из листинга 8.2 в файл lab8-2.asm . . . . .	13
2.14	Сборка программы из файла lab8-2.asm и её запуск . . . . .	14
2.15	Создание третьего файла: lab8-3.asm . . . . .	14
2.16	Запись кода из листинга 8.3 в файл lab8-3.asm . . . . .	15
2.17	Сборка программы из файла lab8-2.asm и её запуск . . . . .	15
2.18	Изменение файла lab8-3.asm . . . . .	16
2.19	Повторная сборка программы из файла lab8-3.asm и её запуск . .	16
3.1	Создание файла самостоятельной работы . . . . .	17
3.2	Код файла самостоятельной работы . . . . .	18
3.3	Код файла самостоятельной работы (продолжение) . . . . .	18
3.4	Сборка и запуск программы первого задания самостоятельной работы, а также результат выполнения . . . . .	19

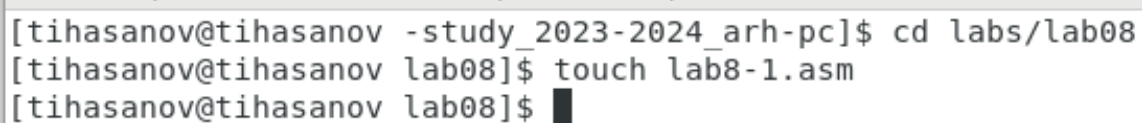
## **Список таблиц**

# 1 Цель работы

Научиться работать с циклами на языке Ассемблера, а также научиться обрабатывать аргументы командной строки

## 2 Выполнение лабораторной работы

Для начала выполнения лабораторной работы перейдем в рабочую директорию и создадим файл lab8-1.asm (рис. 2.1):

A terminal window showing the execution of two commands. The first command is 'cd labs/lab08' and the second is 'touch lab8-1.asm'. The prompt changes from '[tihasanov@tihasanov -study\_2023-2024\_arh-pc]\$' to '[tihasanov@tihasanov lab08]\$' after the first command.

```
[tihasanov@tihasanov -study_2023-2024_arh-pc]$ cd labs/lab08
[tihasanov@tihasanov lab08]$ touch lab8-1.asm
[tihasanov@tihasanov lab08]$
```

Рис. 2.1: Создание файла lab8-1.asm

Далее, запустим Midnight commander (рис. 2.2):

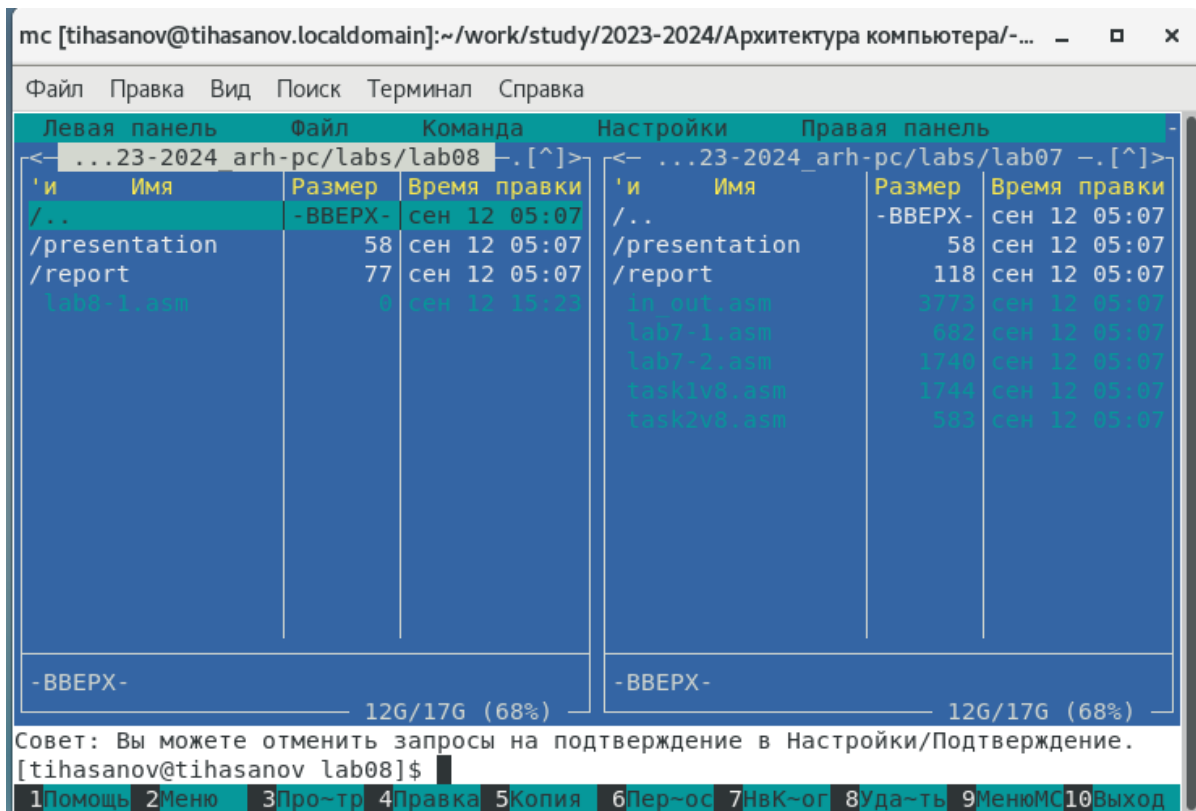


Рис. 2.2: Запуск Midnight commander

Теперь, вставим в ранее созданный файл из листинга 8.1. Он должен запускать цикл и выводить каждую итерацию число, на единицу меньше предыдущего (начинается выводить с числа N) (рис. 2.3):

```

lab8-1.asm      [-M--] 21 L:[ 1+ 0 1/ 28] *(21 / 636b)
#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. 2.3: Вставка кода из файла листинга 8.1

Чтобы собрать код, нужен файл `in_out.asm`. скопируем его из директории прошлой лабораторной работы (рис. 2.4):



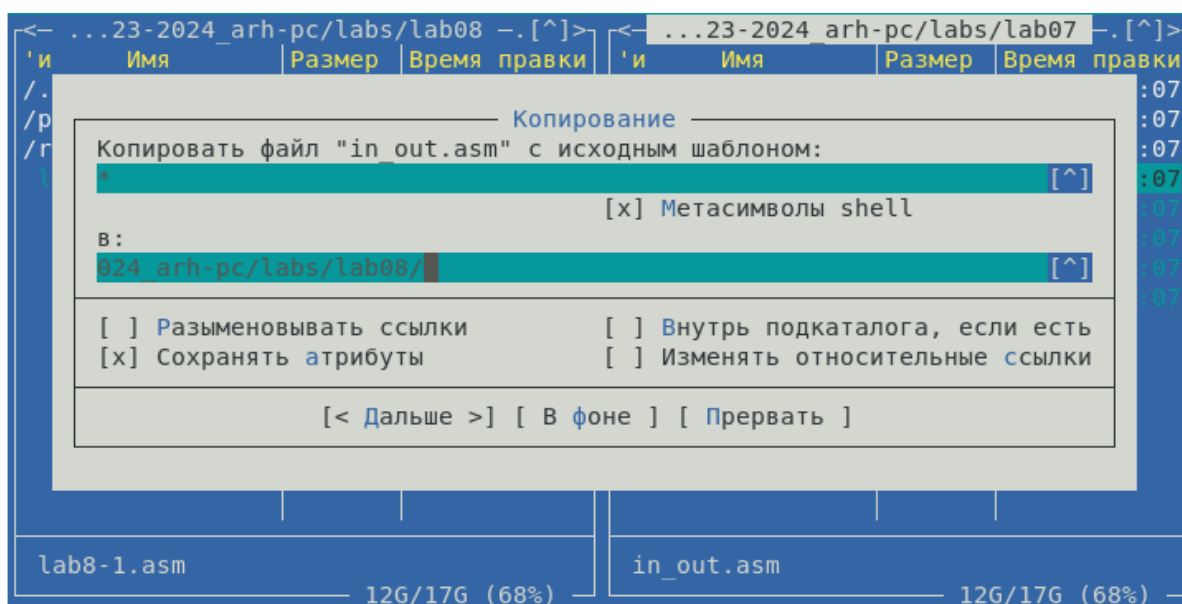


Рис. 2.4: Копирование файла in\_out.asm в рабочую директорию

Теперь соберём программу и посмотрим на результат выполнения (рис. 2.5):

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

Рис. 2.5: Сборка программы из файла lab8-1.asm и её запуск

Как видим, она выводит числа от N до единицы включительно. Теперь попробуем изменить код, чтобы в цикле также отнималась единица у регистра ecx (рис. 2.6):

```

; ----- Организация цикла
mov ecx,[N] ; Счетчик цикла, `ecx=N`
label:
sub ecx,1 ; `ecx=ecx-1`
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. 2.6: Изменение файла lab8-1.asm

Попробуем собрать программу и запустить её (рис. 2.7):

```

[tihasanov@tihasanov lab08]$ nasm -f elf lab8-1.asm
[tihasanov@tihasanov lab08]$ ld -m elf_i386 -o lab8-1 lab8-1.o
[tihasanov@tihasanov lab08]$ ./lab8-1
Введите N: █

```

Рис. 2.7: Повторная сборка программы из файла lab8-1.asm и её запуск

Введём в качестве N число 5 и посмотрим на результат выполнения (рис. 2.8):

```
4294945790
4294945788
4294945786
4294945784
4294945782
4294945780
4294945778
4294945776
4294945774
4294945772
4294945770
4294945768
4294945766
4294945764
4294945762
4294945760
4294945758
4294945756
4294945754
4294945752
4294945750
4294945748
429^C
[tihasanov@tihasanov lab08]$ █
```

Рис. 2.8: Результат вывода

Как видим, цикл выполняется бесконечное количество раз. Это связано с тем, что цикл останавливается в тот момент, когда при проверке `esx` равен 0, но он каждое выполнение цикла уменьшается на 2, из-за чего, в случае нечётного числа, никогда не достигнет нуля. Регистр `esx` меняет своё значение дважды: стандартно -1 после каждой итерации и -1 в теле цикла из-за команды `sub`. Если на вход подать чётное число, цикл прогонится  $N/2$  раз, выводя числа от  $N-1$  до 1 (выводит через одно число) (рис. 2.9):

```
[tihasanov@tihasanov lab08]$ ./lab8-1
Введите N: 12
11
9
7
5
3
1
```

Рис. 2.9: Результат вывода для чётного N

Таким образом, количество итераций цикла не равно N ни при подаче на вход чётного числа, ни при подаче нечётного.

Теперь попробуем изменить программу так, чтобы она сохраняла значение регистра `ecx` в стек (рис. 2.10):

```
; ----- Организация цикла
mov ecx,[N] ; Счетчик цикла, `ecx=N`
label:
push ecx ; Добавление значения ecx в стек
sub ecx,1 ; `ecx=ecx-1`
mov [N],ecx
mov eax,[N]
call iprintLF ; Вывод значения `N`
pop ecx ; Извлечение значения ecx из стека
loop label ; `ecx=ecx-1` и если `ecx` не '0'
; переход на `label`
call quit
```

Рис. 2.10: Редактирование файла lab8-1.asm

Попробуем собрать и запустить программу (рис. 2.11):

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

Рис. 2.11: Повторная сборка программы из файла lab8-1.asm и её запуск

Теперь, программа выводит все числа от N-1 до нуля. Таким образом, число прогонов цикла равно числу N. Создадим второй файл (рис. 2.12):

```
[tihasanov@tihasanov lab08]$ touch lab8-2.asm
[tihasanov@tihasanov lab08]$
```

Рис. 2.12: Создание второго файла: lab8-2.asm

И вставим в него код из файла листинга 8.2 (рис. 2.13):

```
lab8-2.asm      [-M--]  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 sprintLF ; вызываем функцию печати
loop next ; переход к обработке следующего
; аргумента (переход на метку `next`)
_end:
call quit
```

Рис. 2.13: Запись кода из листинга 8.2 в файл lab8-2.asm

Соберём и запустим его, указав некоторые аргументы. Посмотрим на результат (рис. 2.14):

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

Рис. 2.14: Сборка программы из файла lab8-2.asm и её запуск

Как видим, он обработал 4 аргумента. Аргументы разделяются пробелом, либо, когда аргумент содержит в себе пробел, обрамляется в кавычки. Создадим третий файл (рис. 2.15):

```
[tihasanov@tihasanov lab08]$ touch lab8-3.asm
[tihasanov@tihasanov lab08]$ █
```

Рис. 2.15: Создание третьего файла: lab8-3.asm

И вставим в него код из листинга 8.3. Он будет находить сумму всех аргументов (рис. 2.16):

```

lab8-3.asm      [-M--]  6 L:[  1+ 5   6/ 29] *(102 /1427b) 0010 0x00A  [*][X]
%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`

```

Рис. 2.16: Запись кода из листинга 8.3 в файл lab8-3.asm

Теперь соберём программу и запустим её (рис. 2.17):

```

[tihasanov@tihasanov lab08]$ nasm -f elf lab8-3.asm
[tihasanov@tihasanov lab08]$ ld -m elf_i386 -o lab8-3 lab8-3.o
[tihasanov@tihasanov lab08]$ ./lab8-3 12 13 7 10 5
Результат: 47
[tihasanov@tihasanov lab08]$ █

```

Рис. 2.17: Сборка программы из файла lab8-2.asm и её запуск

Как видим, программа действительно выводит сумму всех аргументов. Изменим её так, чтобы она находила не сумму, а произведение всех аргументов (рис. 2.18):

```

; 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 произведений
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
mul esi
mov 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. 2.18: Изменение файла lab8-3.asm

Соберём программу и запустим её (рис. 2.19):

```

[tihasanov@tihasanov lab08]$ nasm -f elf lab8-3.asm
[tihasanov@tihasanov lab08]$ ld -m elf_i386 -o lab8-3 lab8-3.o
[tihasanov@tihasanov lab08]$ ./lab8-3 2 3 4 5
Результат: 120

```

Рис. 2.19: Повторная сборка программы из файла lab8-3.asm и её запуск

Как видим, программа выводит правильный ответ



### 3 Выполнение задания для самостоятельной работы

Для выполнения самостоятельной работы создадим файл в формате .asm (рис. 3.1):

```
[tihasanov@tihasanov lab08]$ touch tasklv8.asm  
[tihasanov@tihasanov lab08]$ █
```

Рис. 3.1: Создание файла самостоятельной работы

В рамках самостоятельной работы необходимо сделать задание под вариантом 8. Так, необходимо сложить результаты выполнения функции  $f(x)=7+2x$  для всех введённых аргументов (рис. 3.2 и рис. 3.3):

```
tasklv8.asm      [-M--]  7 L:[ 1+ 6  7/ 35] *(139 /1683b) 0010 0x00A  [*][X]
%include 'in_out.asm'
SECTION .data
msg db "Результат: ",0
msg2 db "Функция: f(x)=7+2x"
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
```

Рис. 3.2: Код файла самостоятельной работы

```
mov ebx, 2 ; изменяем коэффициент умножения на 2 ( $f(x) = 7 + 2x$ )
mul ebx ; умножаем eax на 2
add eax, 7 ; добавляем 7 к результату
add esi,eax ; добавляем к промежуточной сумме
; след. аргумент `esi=esi+eax`
loop next ; переход к обработке следующего аргумента
_end:
mov eax, msg2
call sprintf
mov eax, msg ; вывод сообщения "Результат: "
call sprintf
mov eax, esi ; записываем сумму в регистр `eax`
call iprintLF ; печать результата
call quit ; завершение программы
```

Рис. 3.3: Код файла самостоятельной работы (продолжение)

Соберём и запустим программу, вводя различные аргументы (рис. 3.4):

```
[tihasanov@tihasanov lab08]$ nasm -f elf task1v8.asm
[tihasanov@tihasanov lab08]$ ld -m elf_i386 -o task1v8 task1v8.o
[tihasanov@tihasanov lab08]$ ./task1v8 1 2 3 4
Функция:  $f(x)=7+2x$ 
Результат: 48
[tihasanov@tihasanov lab08]$ ./task1v8 5 10 15 20
Функция:  $f(x)=7+2x$ 
Результат: 128
[tihasanov@tihasanov lab08]$ ./task1v8 11 22 33 44
Функция:  $f(x)=7+2x$ 
Результат: 248
[tihasanov@tihasanov lab08]$ █
```

Рис. 3.4: Сборка и запуск программы первого задания самостоятельной работы, а также результат выполнения

Пересчитав результат вручную, убеждаемся, что программа работает верно

## 4 Выводы

В результате выполнения лабораторной работы были получены навыки работы с циклами и обработкой аргументов из командной строки. Были написаны программы, использующие все вышеописанные аспекты