

Лабораторная работа №8

Программирование цикла. Обработка аргументов командной строки

Яковлева Дарья Сергеевна

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
3	Выполнение задания для самостоятельной работы	17
4	Выводы	19

Список иллюстраций

2.1	Создание рабочей директории и файла lab8-1.asm	6
2.2	Запуск Midnight commander	6
2.3	Вставка кода из файла листинга 8.1	7
2.4	Копирование файла in_out.asm в рабочую директорию	8
2.5	Сборка программы из файла lab8-1.asm и её запуск	8
2.6	Изменение файла lab8-1.asm	9
2.7	Повторная сборка программы из файла lab8-1.asm и её запуск . . .	9
2.8	Результат вывода	10
2.9	Результат вывода для чётного N	11
2.10	Редактирование файла lab8-1.asm	11
2.11	Повторная сборка программы из файла lab8-1.asm и её запуск . . .	12
2.12	Создание второго файла: lab8-2.asm	12
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	13
2.15	Создание третьего файла: lab8-3.asm	14
2.16	Запись кода из листинга 8.3 в файл lab8-3.asm	14
2.17	Сборка программы из файла lab8-2.asm и её запуск	15
2.18	Изменение файла lab8-3.asm	15
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	Сборка и запуск программы первого задания самостоятельной ра- боты, а также результат выполнения	18

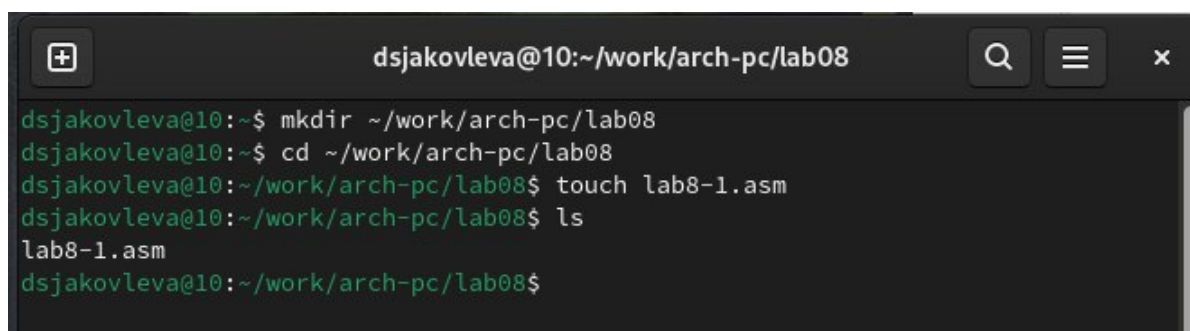
Список таблиц

1 Цель работы

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

2 Выполнение лабораторной работы

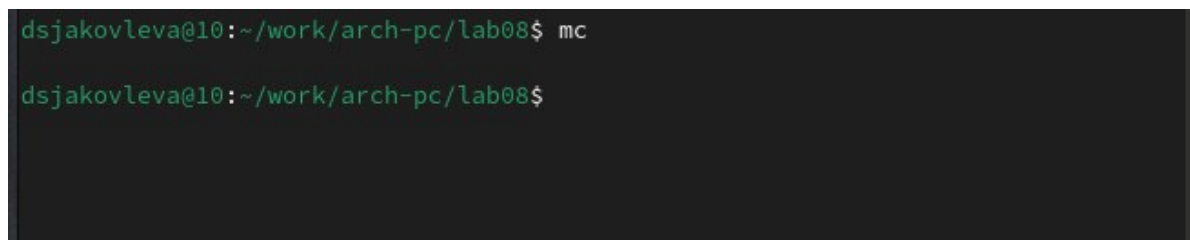
Для начала выполнения лабораторной работы создадим рабочую директорию и файл lab8-1.asm (рис. 2.1):

A terminal window with a dark background. The title bar shows the user 'dsjakovleva@10' and the current directory '~/work/arch-pc/lab08'. The terminal contains the following commands and output:

```
dsjakovleva@10:~$ mkdir ~/work/arch-pc/lab08
dsjakovleva@10:~$ cd ~/work/arch-pc/lab08
dsjakovleva@10:~/work/arch-pc/lab08$ touch lab8-1.asm
dsjakovleva@10:~/work/arch-pc/lab08$ ls
lab8-1.asm
dsjakovleva@10:~/work/arch-pc/lab08$
```

Рис. 2.1: Создание рабочей директории и файла lab8-1.asm

Далее, запустим Midnight commander (рис. 2.2):

A terminal window with a dark background. The title bar shows the user 'dsjakovleva@10' and the current directory '~/work/arch-pc/lab08'. The terminal contains the following commands and output:

```
dsjakovleva@10:~/work/arch-pc/lab08$ mc
dsjakovleva@10:~/work/arch-pc/lab08$
```

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):

```
mc [dsjakovleva@10.0.2.15]:~/work/arch-pc/la...
mc [dsjakovleva@10.0.2.1... x mc [dsjakovleva@10.0.2.1... x
/home/dsjakovleva/work/arch-pc/lab08/lab8-1.asm Изменён
-----
; Программа вывода значений регистра 'ecx'
-----
%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

^G Справка ^O Записать ^W Поиск ^K Вырезать ^T Выполнить
^X Выход ^R ЧитФайл ^\ Замена ^U Вставить ^J Выровнять
```

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):

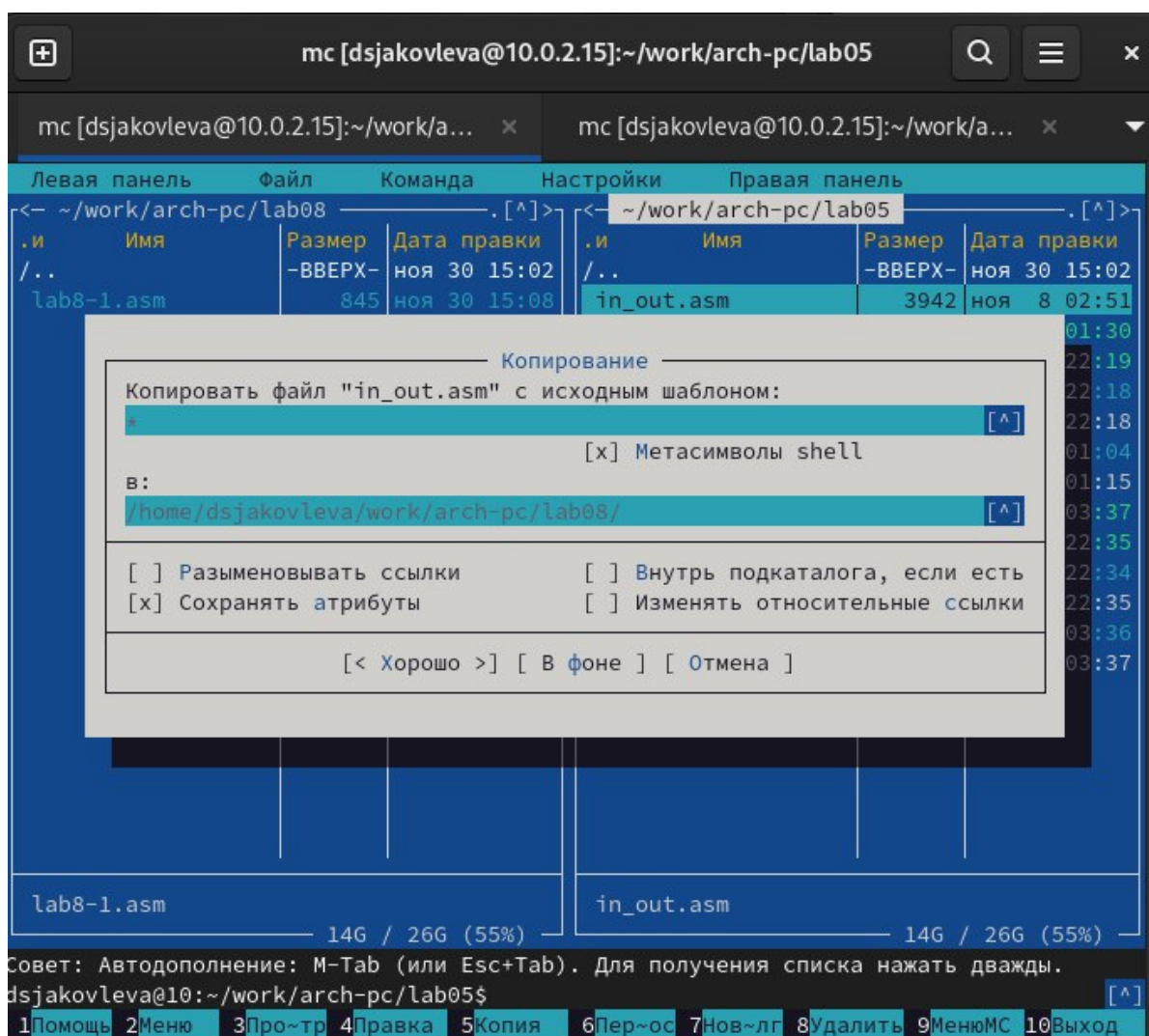


Рис. 2.4: Копирование файла `in_out.asm` в рабочую директорию

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

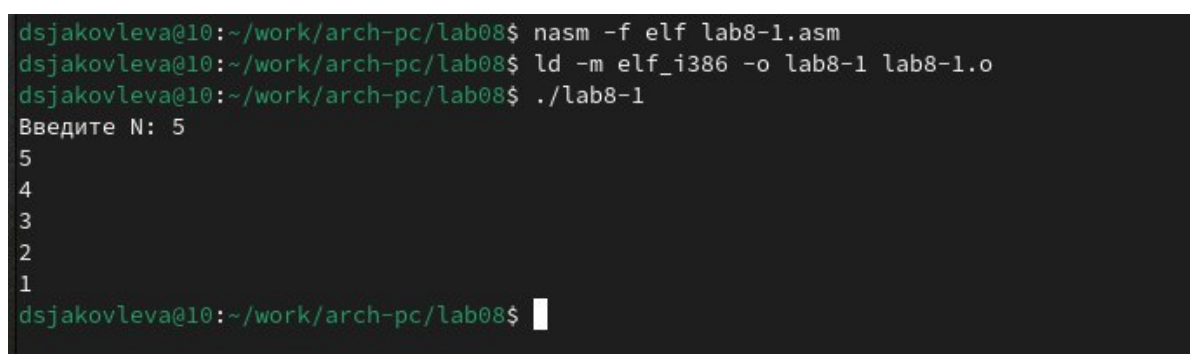


Рис. 2.5: Сборка программы из файла `lab8-1.asm` и её запуск

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):

```
_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
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
1Помощь 2Сохран 3Блок 4Замена 5Копия 6Пер~ть 7Поиск 8Удалить 9МенюМС 10Выход
```

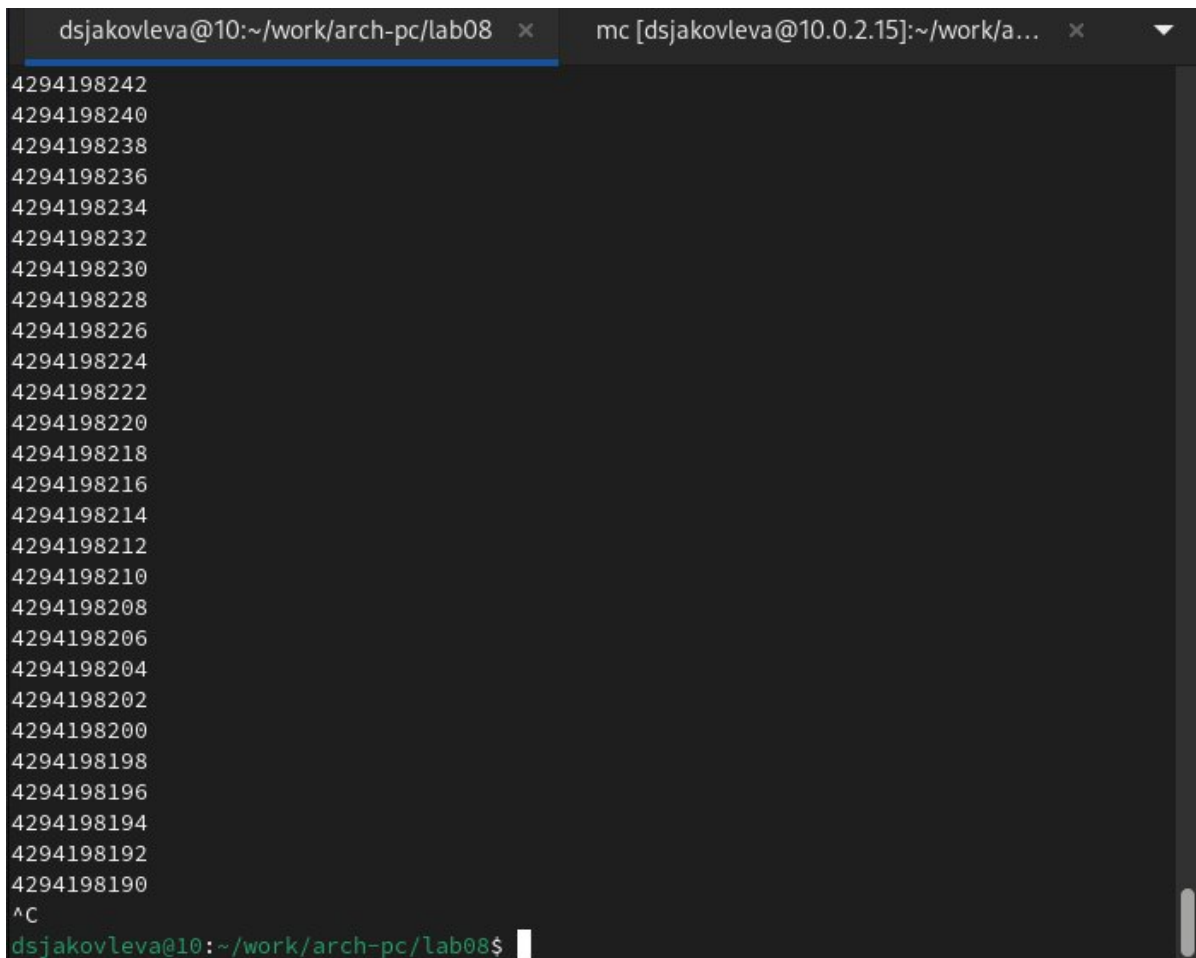
Рис. 2.6: Изменение файла lab8-1.asm

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

```
dsjakovleva@10:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
dsjakovleva@10:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
dsjakovleva@10:~/work/arch-pc/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):



```
dsjakovleva@10:~/work/arch-pc/lab08 x mc [dsjakovleva@10.0.2.15]:~/work/a... x
4294198242
4294198240
4294198238
4294198236
4294198234
4294198232
4294198230
4294198228
4294198226
4294198224
4294198222
4294198220
4294198218
4294198216
4294198214
4294198212
4294198210
4294198208
4294198206
4294198204
4294198202
4294198200
4294198198
4294198196
4294198194
4294198192
4294198190
^C
dsjakovleva@10:~/work/arch-pc/lab08$
```

Рис. 2.8: Результат вывода

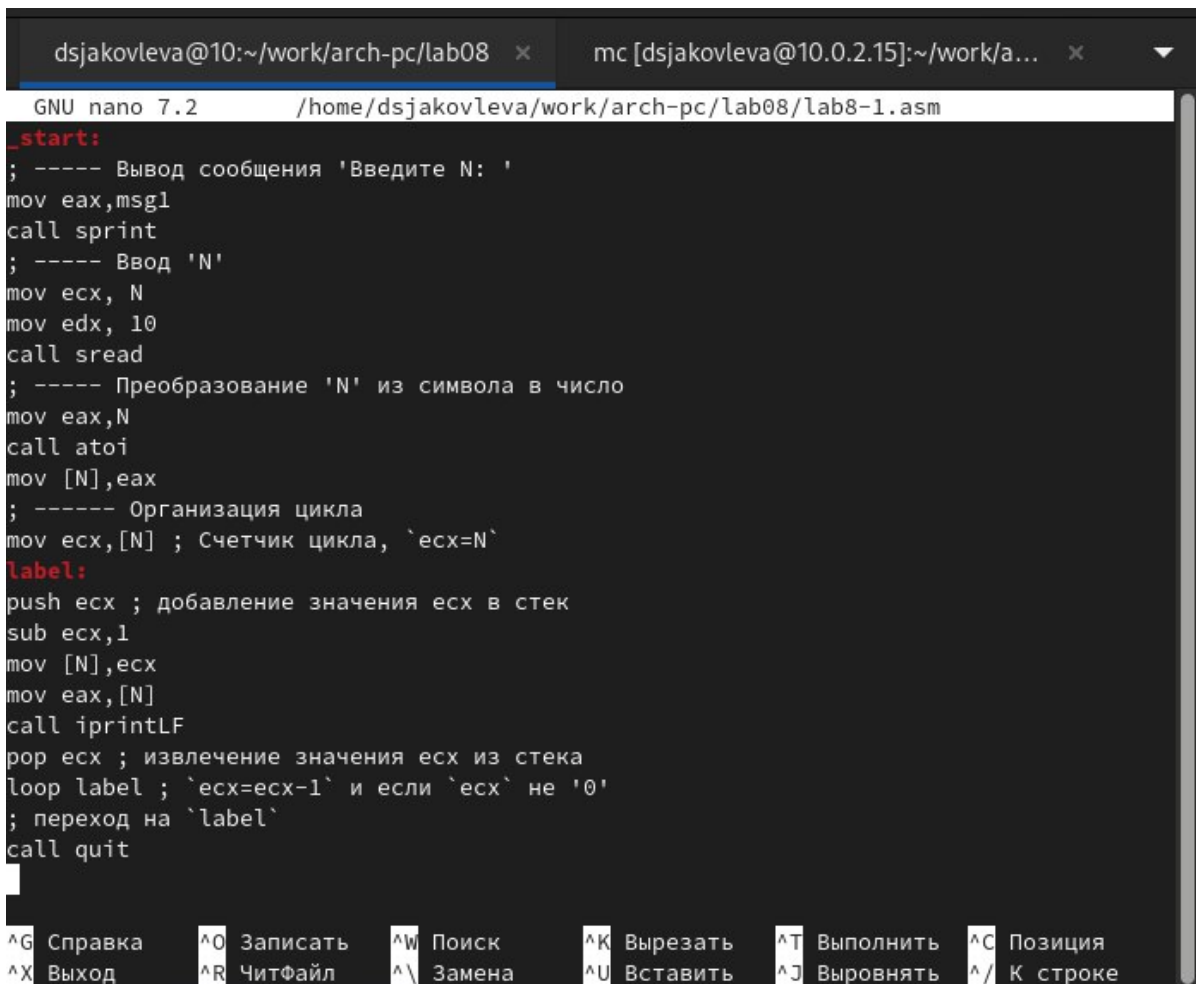
Как видим, цикл выполняется бесконечное количество раз. Это связано с тем, что цикл останавливается в тот момент, когда при проверке `esx` равен 0, но он каждое выполнение цикла уменьшается на 2, из-за чего, в случае нечётного числа, никогда не достигнет нуля. Регистр `esx` меняет своё значение дважды: 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):

```
dsjakovleva@10:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 12
11
9
7
5
3
1
dsjakovleva@10:~/work/arch-pc/lab08$
```

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):



```
dsjakovleva@10:~/work/arch-pc/lab08 x mc [dsjakovleva@10.0.2.15]:~/work/a... x
GNU nano 7.2 /home/dsjakovleva/work/arch-pc/lab08/lab8-1.asm
_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 ; `ecx=ecx-1` и если `ecx` не '0'
; переход на `label`
call quit

```

^G Справка ^O Записать ^W Поиск ^K Вырезать ^T Выполнить ^C Позиция
^X Выход ^R ЧитФайл ^\ Замена ^U Вставить ^J Выводить ^/ К строке

Рис. 2.10: Редактирование файла lab8-1.asm

Попробуем собрать и запустить программу (рис. 2.11):

```
dsjakovleva@10:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
dsjakovleva@10:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
dsjakovleva@10:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 5
4
3
2
1
0
dsjakovleva@10:~/work/arch-pc/lab08$
```

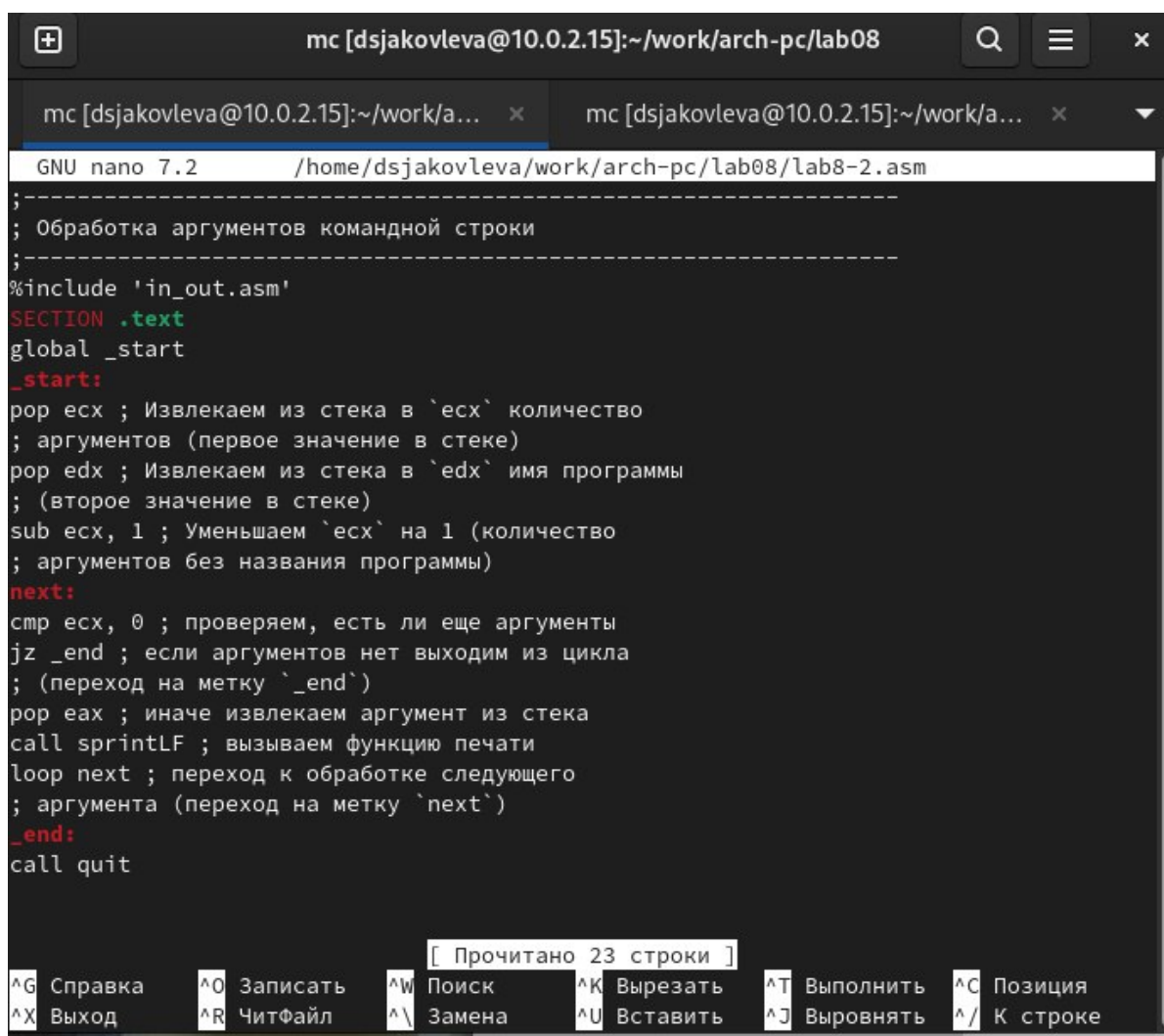
Рис. 2.11: Повторная сборка программы из файла lab8-1.asm и её запуск

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

```
dsjakovleva@10:~/work/arch-pc/lab08$ touch lab8-2.asm
dsjakovleva@10:~/work/arch-pc/lab08$ ls
in_out.asm lab8-1 lab8-1.asm lab8-1.o lab8-2.asm
dsjakovleva@10:~/work/arch-pc/lab08$
```

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

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

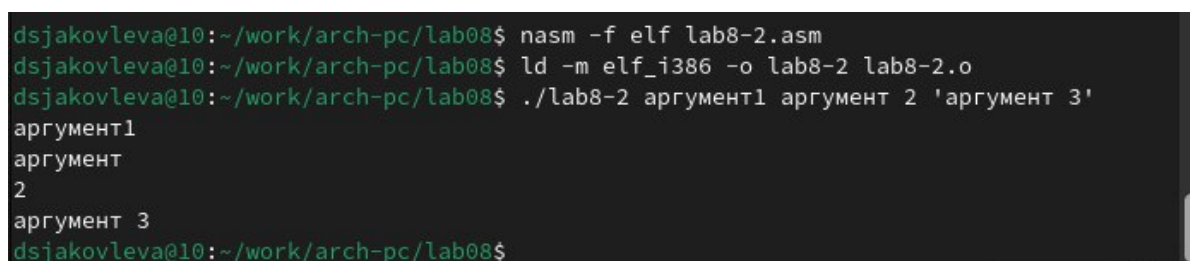


```
mc [dsjakovleva@10.0.2.15]:~/work/arch-pc/lab08
GNU nano 7.2 /home/dsjakovleva/work/arch-pc/lab08/lab8-2.asm
;-----
; Обработка аргументов командной строки
;-----
%include 'in_out.asm'
SECTION .text
global _start
_start:
por ecx ; Извлекаем из стека в `ecx` количество
; аргументов (первое значение в стеке)
por 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`)
por 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

[ Прочитано 23 строки ]
^G Справка ^O Записать ^W Поиск ^K Вырезать ^T Выполнить ^C Позиция
^X Выход ^R ЧитФайл ^_ Замена ^U Вставить ^J Выводить ^_/ К строке
```

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):



```
dsjakovleva@10:~/work/arch-pc/lab08$ nasm -f elf lab8-2.asm
dsjakovleva@10:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-2 lab8-2.o
dsjakovleva@10:~/work/arch-pc/lab08$ ./lab8-2 аргумент1 аргумент 2 'аргумент 3'
аргумент1
аргумент
2
аргумент 3
dsjakovleva@10:~/work/arch-pc/lab08$
```

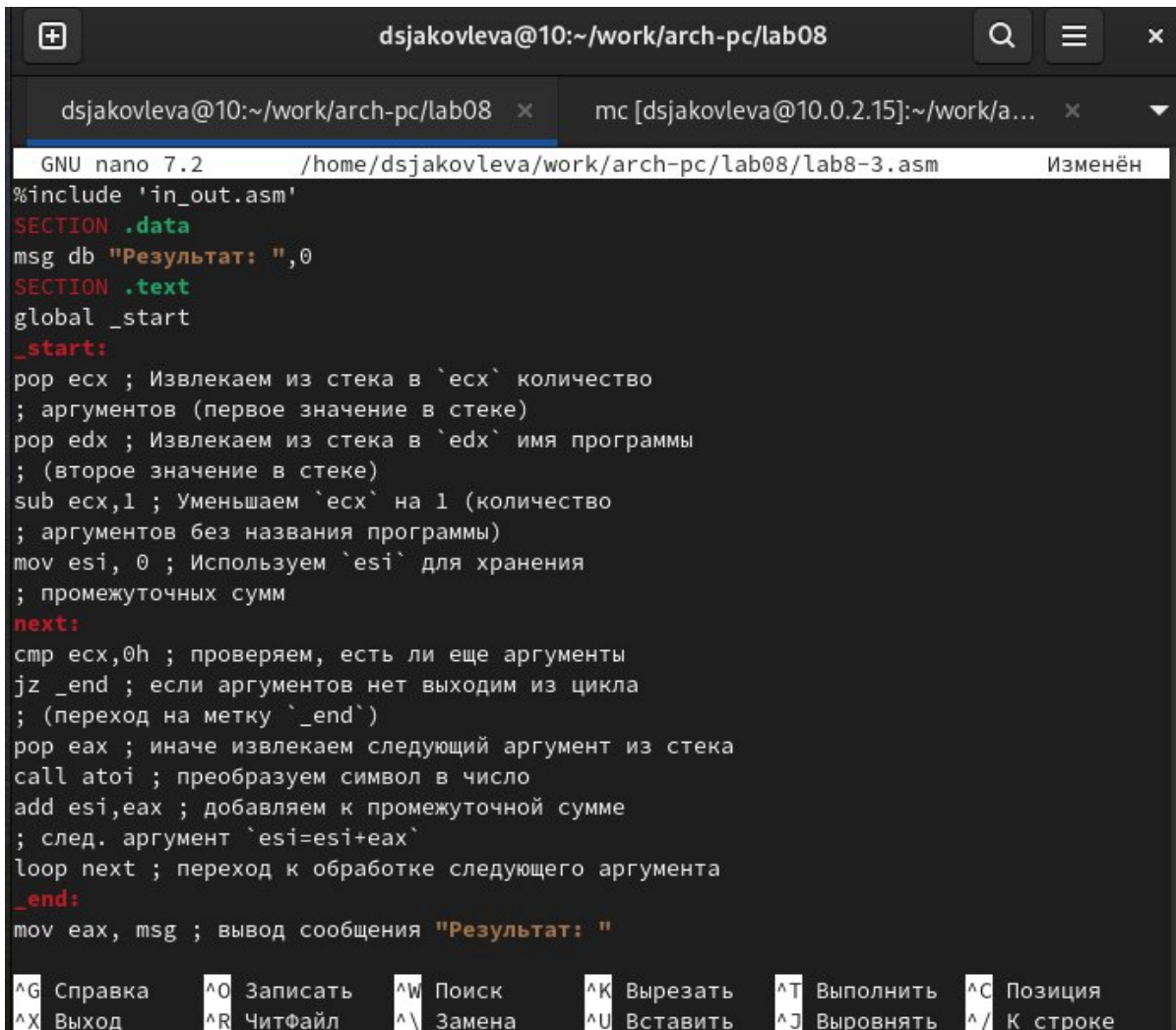
Рис. 2.14: Сборка программы из файла lab8-2.asm и её запуск

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):

```
dsjakovleva@10:~/work/arch-pc/lab08$ touch lab8-3.asm
dsjakovleva@10:~/work/arch-pc/lab08$ ls
in_out.asm  lab8-1  lab8-1.asm  lab8-1.o  lab8-2  lab8-2.asm  lab8-2.o  lab8-3.asm
dsjakovleva@10:~/work/arch-pc/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):



```
GNU nano 7.2 /home/dsjakovleva/work/arch-pc/lab08/lab8-3.asm Изменён
%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: "
```

[^]G Справка [^]O Записать [^]W Поиск [^]K Вырезать [^]T Выполнить [^]C Позиция
[^]X Выход [^]R ЧитФайл [^]\ Замена [^]U Вставить [^]J Выровнять [^]/ К строке

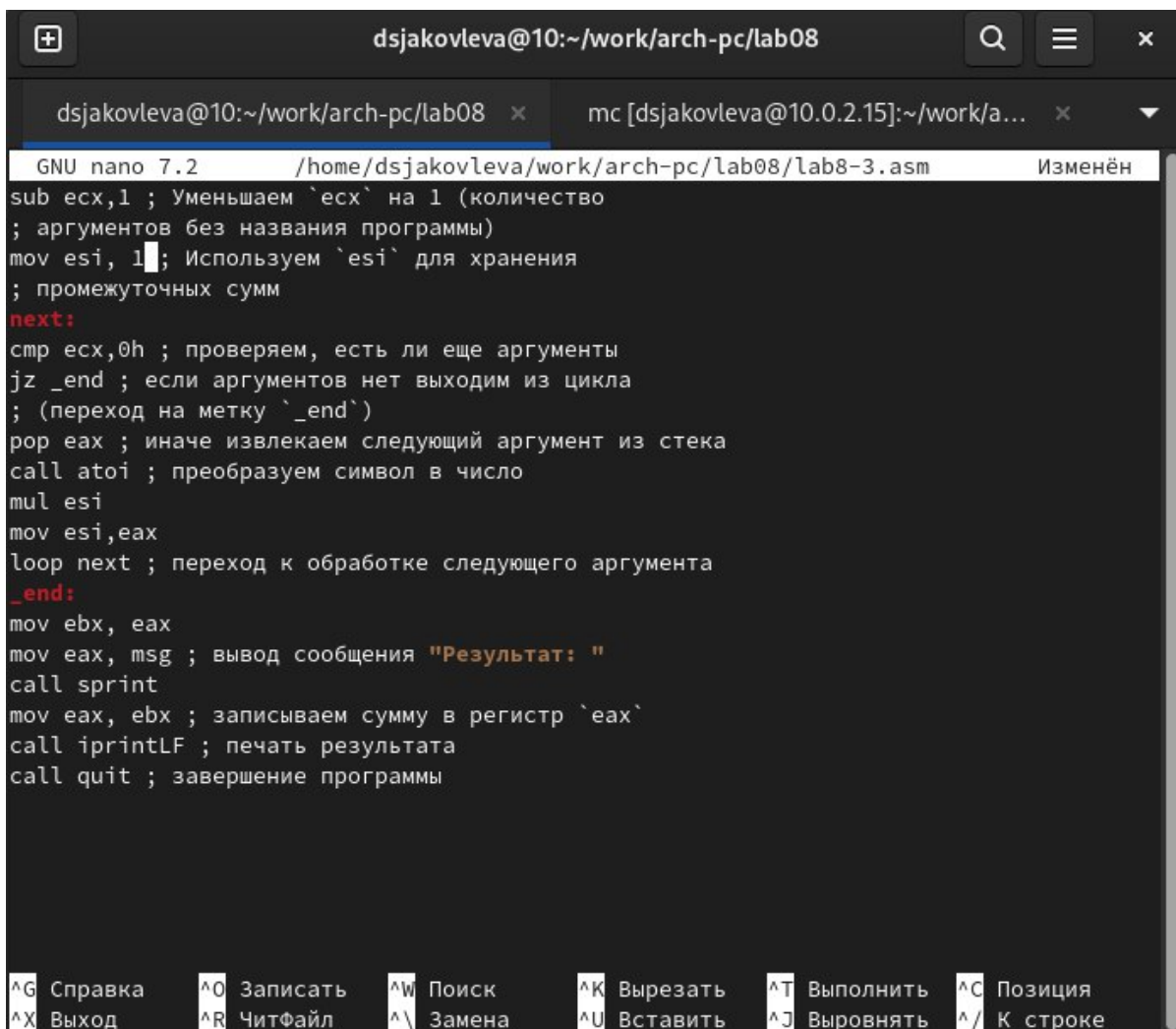
Рис. 2.16: Запись кода из листинга 8.3 в файл lab8-3.asm

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

```
dsjakovleva@10:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
dsjakovleva@10:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
dsjakovleva@10:~/work/arch-pc/lab08$ ./lab8-3 12 13 7 10 5
Результат: 47
dsjakovleva@10:~/work/arch-pc/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):



```
GNU nano 7.2 /home/dsjakovleva/work/arch-pc/lab08/lab8-3.asm Изменён
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
mul esi
mov esi,eax
loop next ; переход к обработке следующего аргумента
_end:
mov ebx, eax
mov eax, msg ; вывод сообщения "Результат: "
call sprint
mov eax, ebx ; записываем сумму в регистр `eax`
call iprintLF ; печать результата
call quit ; завершение программы

^G Справка      ^O Записать     ^W Поиск        ^K Вырезать     ^T Выполнить    ^С Позиция
^X Выход        ^R ЧитФайл     ^\ Замена       ^U Вставить     ^J Выровнять    ^/ К строке
```

Рис. 2.18: Изменение файла lab8-3.asm

Соберём программу и запустим её (рис. 2.19):

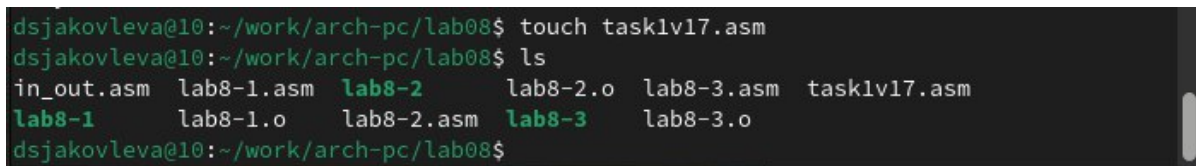
```
dsjakovleva@10:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
dsjakovleva@10:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
dsjakovleva@10:~/work/arch-pc/lab08$ ./lab8-3 2 3 4 5
Результат: 120
dsjakovleva@10:~/work/arch-pc/lab08$
```

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):

A screenshot of a terminal window with a dark background. The prompt is 'dsjakovleva@10:~/work/arch-pc/lab08\$'. The first command is 'touch tasklv17.asm'. The second command is 'ls', which lists the following files: 'in_out.asm', 'lab8-1.asm', 'lab8-2', 'lab8-2.o', 'lab8-3.asm', and 'tasklv17.asm'. The third command is another 'ls', which lists: 'lab8-1', 'lab8-1.o', 'lab8-2.asm', 'lab8-3', and 'lab8-3.o'. The prompt returns at the end of the second 'ls' command.

```
dsjakovleva@10:~/work/arch-pc/lab08$ touch tasklv17.asm
dsjakovleva@10:~/work/arch-pc/lab08$ ls
in_out.asm  lab8-1.asm  lab8-2      lab8-2.o  lab8-3.asm  tasklv17.asm
lab8-1      lab8-1.o    lab8-2.asm  lab8-3    lab8-3.o
dsjakovleva@10:~/work/arch-pc/lab08$
```

Рис. 3.1: Создание файла самостоятельной работы

В рамках самостоятельной работы необходимо сделать задание под вариантом 17. Так, необходимо сложить результаты выполнения функции $f(x)=10(\pi - 1)$ для всех введённых аргументов (рис. 3.2):

```

%include 'in_out.asm'
SECTION .data
msg db "Результат: ",0
msg2 db "Функция: f(x)=10(x - 1)",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
    mov ebx, 10
    sub eax, 1
    mul ebx
    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 iprintf ; печать результата
    call quit ; завершение программы

```

Рис. 3.2: Код файла самостоятельной работы

Соберём и запустим программу, вводя различные аргументы (рис. 3.3):

```

dsjakovleva@10:~/work/arch-pc/lab08$ nasm -f elf tasklv17.asm
dsjakovleva@10:~/work/arch-pc/lab08$ ld -m elf_i386 -o tasklv17 tasklv17.o
dsjakovleva@10:~/work/arch-pc/lab08$ ./tasklv17 1 2 3 4
Функция: f(x)=10(x - 1)
Результат: 60
dsjakovleva@10:~/work/arch-pc/lab08$ ./tasklv17 5 10 15 20
Функция: f(x)=10(x - 1)
Результат: 460
dsjakovleva@10:~/work/arch-pc/lab08$ ./tasklv17 5 3 6 4 54 6
Функция: f(x)=10(x - 1)
Результат: 720
dsjakovleva@10:~/work/arch-pc/lab08$

```

Рис. 3.3: 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.