

Лабораторная работа №8

Дисциплина: Архитектура компьютера

Малюга Валерия Васильевна

Содержание

1	Цель работы	4
2	Задание	5
3	Теоретическое введение	6
4	Выполнение лабораторной работы	8
4.1	Реализация циклов в NASM	8
4.2	Обработка аргументов командной строки	12
4.3	Задание для самостоятельной работы	15
5	Выводы	17

Список иллюстраций

4.1	Создание каталога и файла lab8-1.asm	8
4.2	Создание исполняемого файла и его запуск	9
4.3	Изменение кода и запуск программы	10
4.4	Изменение кода и запуск программы	11
4.5	Создание исполняемого файла и его запуск	12
4.6	Создание исполняемого файла и его запуск	13
4.7	Изменение кода и запуск программы	14
4.8	Запуск исполняемого файла и проверка его работы	15

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 Теоретическое введение

Стек — это структура данных, организованная по принципу LIFO («Last In — First Out») или «последним пришёл — первым ушёл»). Стек является частью архитектуры процессора и реализован на аппаратном уровне. Для работы со стеком в процессоре есть специальные регистры (ss, bp, sp) и команды. Основной функцией стека является функция сохранения адресов возврата и передачи аргументов при вызове процедур. Кроме того, в нём выделяется память для локальных переменных и могут временно храниться значения регистров.

Стек имеет вершину, адрес последнего добавленного элемента, который хранится в регистре esp (указатель стека). Противоположный конец стека называется дном. Значение, помещённое в стек последним, извлекается первым. При помещении значения в стек указатель стека уменьшается, а при извлечении — увеличивается.

Команда push размещает значение в стеке, т.е. помещает значение в ячейку памяти, на которую указывает регистр esp, после этого значение регистра esp увеличивается на 4. Данная команда имеет один операнд — значение, которое необходимо поместить в стек.

Команда pop извлекает значение из стека, т.е. извлекает значение из ячейки памяти, на которую указывает регистр esp, после этого уменьшает значение регистра esp на 4. У этой команды также один операнд, который может быть регистром или переменной в памяти. Нужно помнить, что извлечённый из стека элемент не стирается из памяти и остаётся как “мусор”, который будет перезаписан при записи нового значения в стек.

Для организации циклов существуют специальные инструкции. Для всех ин-

струкций максимальное количество проходов задаётся в регистре esx. Наиболее простой является инструкция loor. Она позволяет организовать безусловный цикл.

4 Выполнение лабораторной работы

4.1 Реализация циклов в NASM

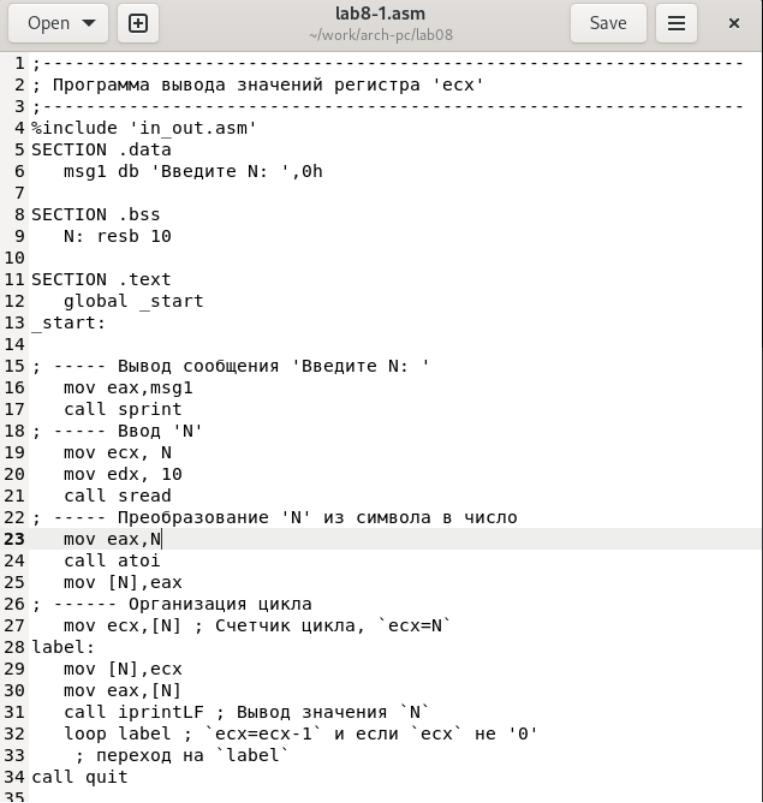
Создала каталог для программ лабораторной работы № 8, перешла в него и создала файл lab8-1.asm (рис. 4.1).

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

Рис. 4.1: Создание каталога и файла lab8-1.asm

Ввела в файл lab8-1.asm текст программы из листинга 8.1. Создала исполняемый файл и проверила его работу (рис. 4.2). Программа работает корректно, выводит числа от N до 1 включительно.

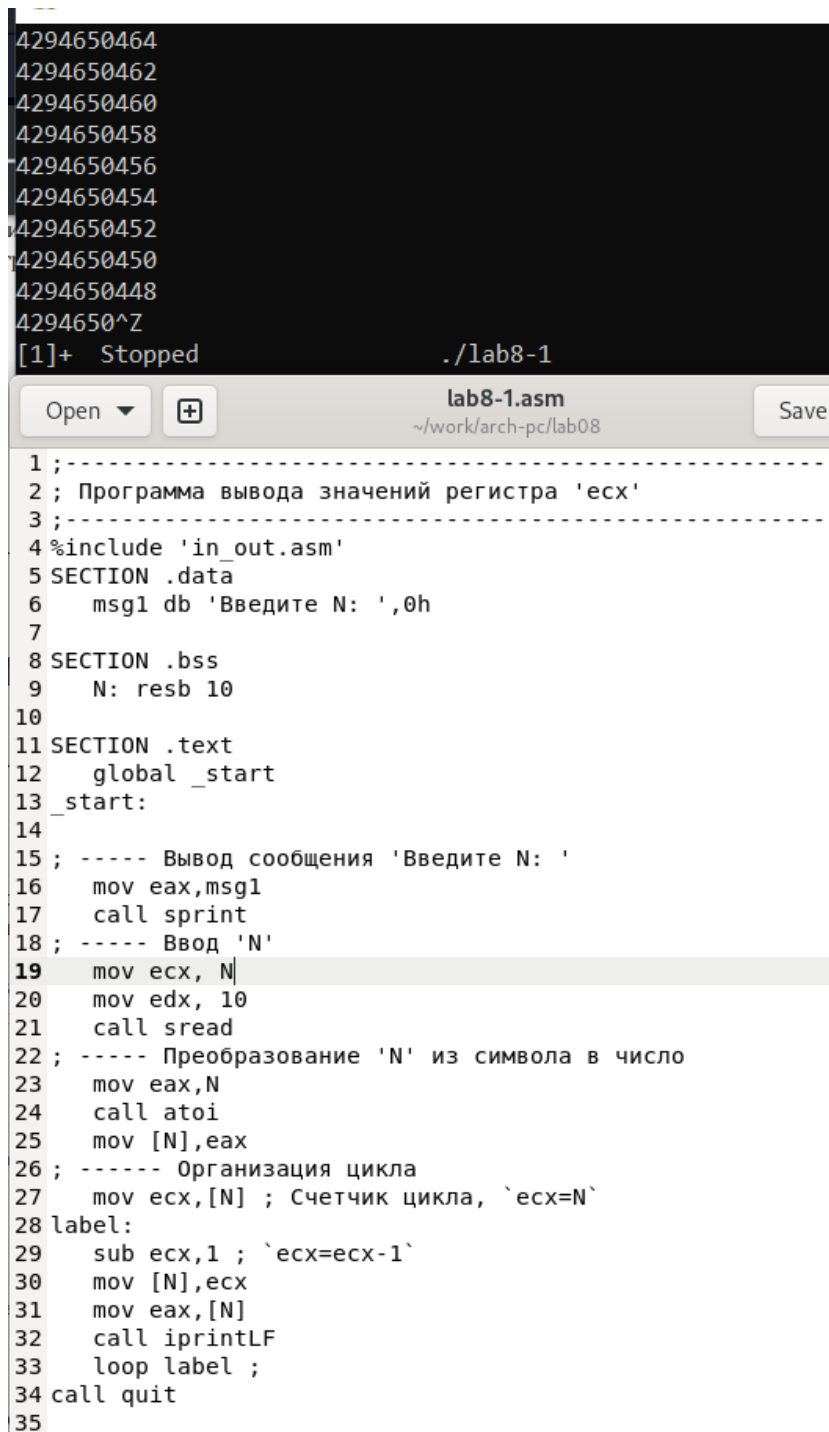

```
vvmalyuga@Malyuga:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
vvmalyuga@Malyuga:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
vvmalyuga@Malyuga:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 5
5
4
3
2
1
```



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

Рис. 4.2: Создание исполняемого файла и его запуск

Изменила текст программы, добавив изменение значение регистра `ecx` в цикле. Создала исполняемый файл и проверила его работу (рис. 4.3). В данном случае число проходов цикла не соответствует введенному с клавиатуры значению.



The image shows a debugger window with a memory dump at the top and a code editor below. The memory dump lists addresses from 4294650464 to 4294650448, ending with 4294650^Z and a status bar indicating '[1]+ Stopped ./lab8-1'. The code editor is titled 'lab8-1.asm' and shows assembly code for a program that reads a number 'N' and prints it. Line 19, 'mov ecx, N', is highlighted.

```
4294650464
4294650462
4294650460
4294650458
4294650456
4294650454
4294650452
4294650450
4294650448
4294650^Z
[1]+ Stopped ./lab8-1

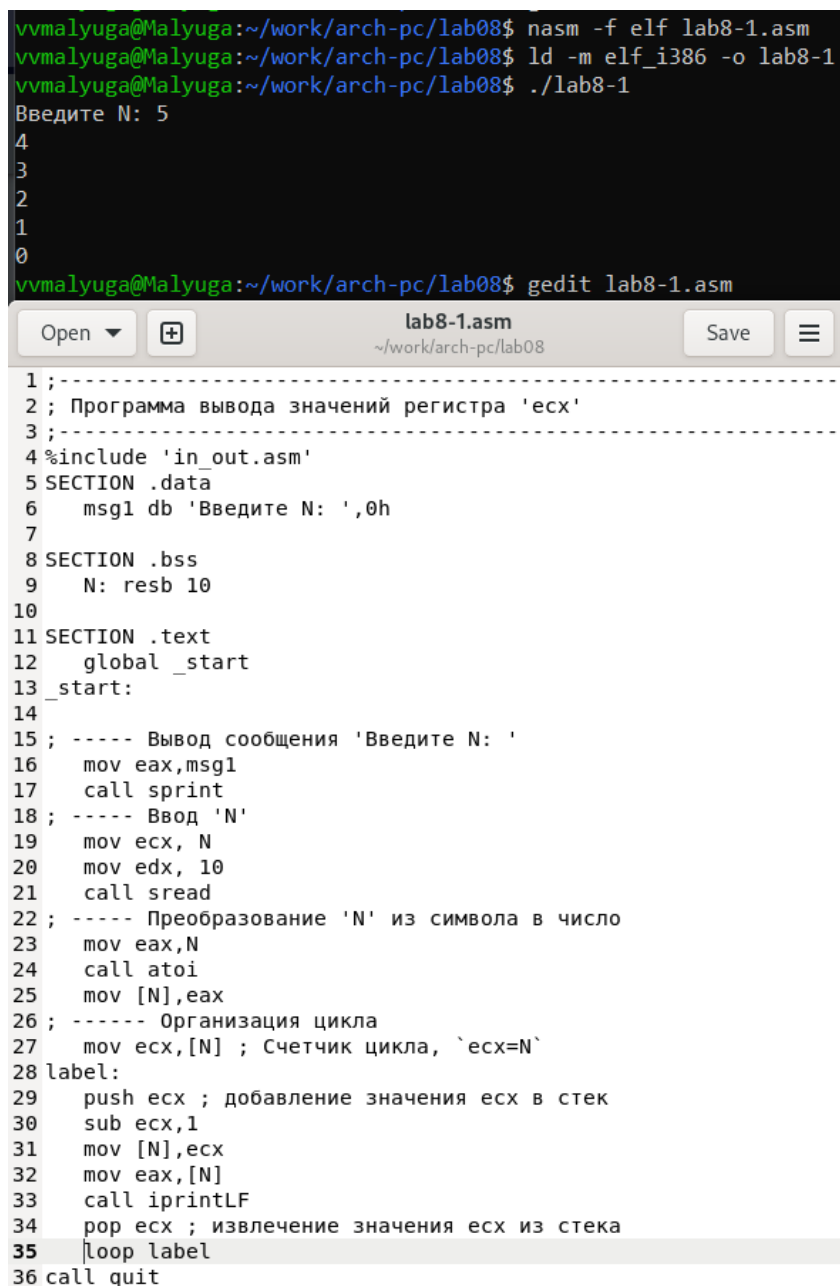
lab8-1.asm
~/work/arch-pc/lab08

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

Рис. 4.3: Изменение кода и запуск программы

Внесла изменения в текст программы, добавив команды `push` и `pop` (добавления в стек и извлечения из стека) для сохранения значения счетчика цикла `loop` (рис.

4.4). Создала исполняемый файл и проверила его работу. В данном случае число проходов цикла соответствует введенному с клавиатуры значению и выводит числа от N-1 до 0 включительно.



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

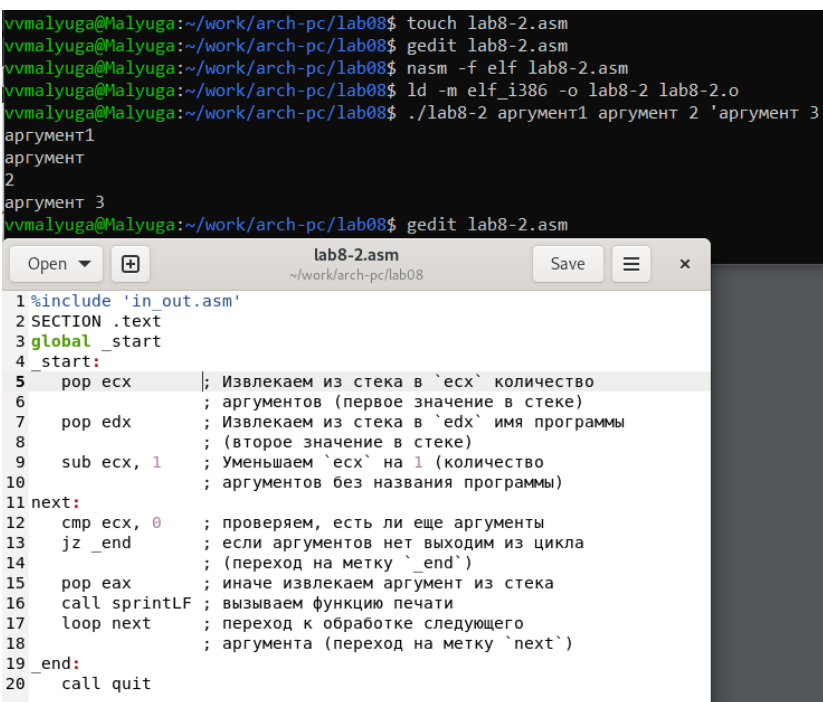
lab8-1.asm
~/work/arch-pc/lab08

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

Рис. 4.4: Изменение кода и запуск программы

4.2 Обработка аргументов командной строки

Создала файл lab8-2.asm в каталоге work/arch-pc/lab08 и ввела в него текст программы из листинга 8.2. Создала исполняемый файл и запустила его, указав аргументы: аргумент1 аргумент 2 'аргумент 3' (рис. 4.5). В данном случае программой было обработано 4 аргумента.



The image shows a terminal window and a code editor. The terminal window displays the following commands and output:

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

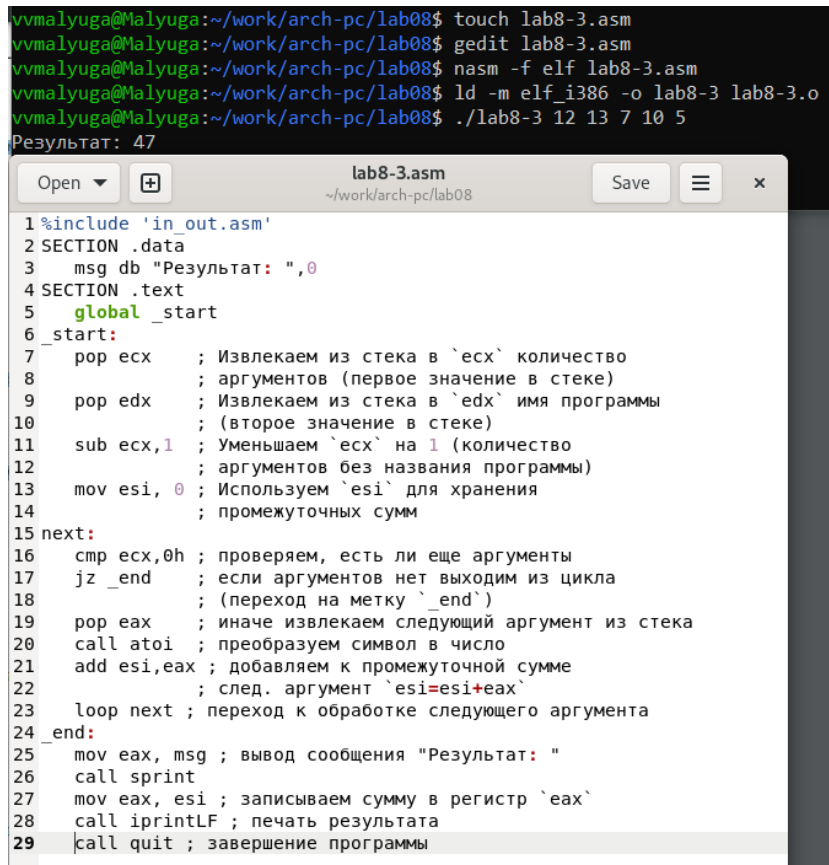
The code editor shows the following assembly code:

```
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. 4.5: Создание исполняемого файла и его запуск

Создала файл lab8-3.asm в каталоге work/arch-pc/lab08 и ввела в него текст программы из листинга 8.3. Создала исполняемый файл и запустила его, указав аргументы: 12 13 7 10 5 (рис. 4.6).

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



```
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. 4.6: Создание исполняемого файла и его запуск

Изменила текст программы из листинга 8.3 для вычисления произведения аргументов командной строки (рис. 4.7). Для этого в 13 строке изменила начальное значение счетчика на 1, в 21 строке изменила add на imul.

Прилагаю код:

```
%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
```

```

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 промежуточного результата

```

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
imul esi, eax ; умножаем промежуточный результат на след. аргумент `esi=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

```

```

vvmalyuga@Malyuga:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
vvmalyuga@Malyuga:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
vvmalyuga@Malyuga:~/work/arch-pc/lab08$ ./lab8-3 1 2 3 4
Результат: 24

```

```

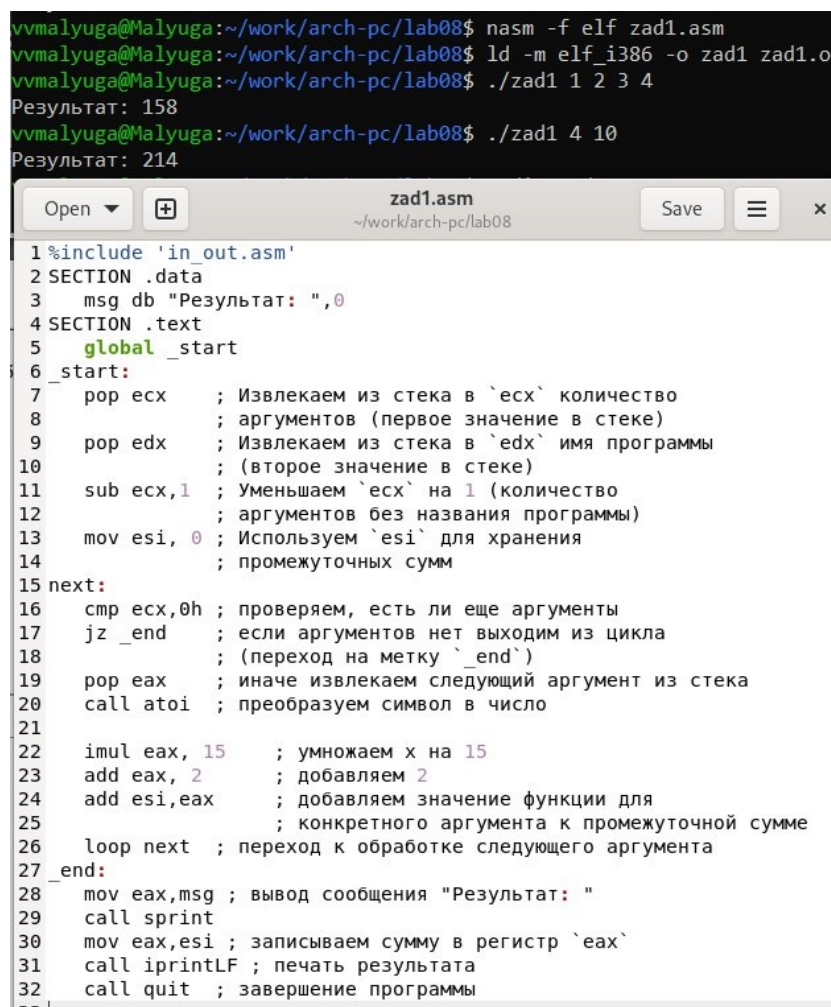
1 %include 'in_out.asm'
2 SECTION .data
3   msg db "Результат: ",0
4 SECTION .text
5   global _start
6 _start:
7   pop ecx      ; Извлекаем из стека в `ecx` количество аргументов (первое значение в стеке)
8   pop edx      ; Извлекаем из стека в `edx` имя программы (второе значение в стеке)
9   sub ecx,1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

```

Рис. 4.7: Изменение кода и запуск программы

4.3 Задание для самостоятельной работы

Написала программу, которая находит сумму значений функции $f(x) = 15x + 2$ (Вариант 11) для $x = x_1, x_2, \dots, x_n$. Создала исполняемый файл и проверила его работу на нескольких наборах $x = x_1, x_2, \dots, x_n$ (рис. 4.8).



The image shows a terminal window and a code editor. The terminal window displays the following commands and output:

```
vvmalyuga@Malyuga:~/work/arch-pc/lab08$ nasm -f elf zad1.asm
vvmalyuga@Malyuga:~/work/arch-pc/lab08$ ld -m elf_i386 -o zad1 zad1.o
vvmalyuga@Malyuga:~/work/arch-pc/lab08$ ./zad1 1 2 3 4
Результат: 158
vvmalyuga@Malyuga:~/work/arch-pc/lab08$ ./zad1 4 10
Результат: 214
```

The code editor shows the assembly code for `zad1.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
22     imul eax, 15  ; умножаем x на 15
23     add eax, 2    ; добавляем 2
24     add esi,eax   ; добавляем значение функции для
25                 ; конкретного аргумента к промежуточной сумме
26     loop next    ; переход к обработке следующего аргумента
27 _end:
28     mov eax,msg  ; вывод сообщения "Результат: "
29     call sprint
30     mov eax,esi  ; записываем сумму в регистр `eax`
31     call iprintLF ; печать результата
32     call quit    ; завершение программы
33
```

Рис. 4.8: Запуск исполняемого файла и проверка его работы

Прилагаю код:

```
%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

imul eax, 15 ; умножаем x на 15
add eax, 2   ; добавляем 2
add esi,eax  ; добавляем значение функции для
               ; конкретного аргумента 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
```


5 Выводы

Благодаря данной лабораторной работе я приобрела навыки написания программ использованием циклов и обработкой аргументов командной строки, что поможет мне при выполнении последующих лабораторных работ.