

Отчёт по лабораторной работе №8

Дисциплина: Архитектура Компьютера

Дарина Андреевна Куокконен

Содержание

1	Цель работы	4
2	Задание	5
3	Теоретическое введение	6
4	Выполнение лабораторной работы	8
5	Выводы	18
	Список литературы	19

Список иллюстраций

4.1	Работа с директориями и создание файла	8
4.2	Редактирование файла	9
4.3	Копирование файла	9
4.4	Подготовка и исполнение файла	10
4.5	Редактирование файла	10
4.6	Подготовка и запуск исполняемого файла	11
4.7	Создание, редактирование файла	11
4.8	Создание и запуск исполняемого файла	12
4.9	Редактирование файла	12
4.10	Создание файла, открытие его в режиме правки, компиляция и обработка исполняемого файла	13
4.11	Запуск исполняемого файла	13
4.12	Редактирование файла	13
4.13	Компиляция и обработка исполняемого файла	14
4.14	Открытие листинга, редактирование файла	14
4.15	Компиляция и запуск исполняемого файла	15
4.16	Открытие и редактирование файла	15
4.17	Компиляция, обработка и запуск исполняемого файла	16

1 Цель работы

Цель данной лабораторной работы - приобретение практического опыта в написании программ с использованием циклов и обработкой аргументов командной строки.

2 Задание

1. Реализация циклов в NASM.
2. Обработка аргументов командной строки.
3. Выполнение заданий 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. Для стека существует две основные операции:

- добавление элемента в вершину стека (push);
- извлечение элемента из вершины стека (pop).

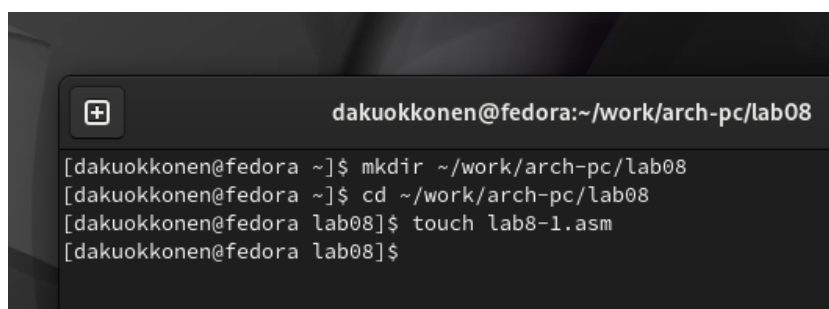
Команда push размещает значение в стеке, т.е. помещает значение в ячейку памяти, на которую указывает регистр esp, после этого значение регистра esp увеличивается на 4. Данная команда имеет один операнд — значение, которое необходимо поместить в стек. Существует ещё две команды для добавления значений в стек. Это команда pusha, которая помещает в стек содержимое всех регистров общего назначения в следующем порядке: ax, cx, dx, bx, sp, bp, si, di. А также команда pushf, которая служит для перемещения в стек содержимого регистра флагов. Обе эти команды не имеют операндов. 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 инструкций максимальное количество проходов задаётся в регистре `ecx`. Наиболее простой является инструкция `loop`. Она позволяет организовать безусловный цикл. Инструкция `loop` выполняется в два этапа. Сначала из регистра `ecx` вычитается единица и его значение сравнивается с нулём. Если регистр не равен нулю, то выполняется переход к указанной метке. Иначе переход не выполняется и управление передаётся команде, которая следует сразу после команды `loop`.

4 Выполнение лабораторной работы

4.1) Реализация циклов в NASM.

С помощью утилиты `mkdir` создаю директорию `lab08` для выполнения соответствующей лабораторной работы. Перехожу в созданный каталог с помощью утилиты `cd`. С помощью `touch` создаю файл `lab8-1.asm`. (рис. 4.1).

A screenshot of a terminal window with a dark background. The title bar shows a window icon and the text "dakuokkonen@fedora:~/work/arch-pc/lab08". The terminal content shows four lines of commands and their outputs: 1. "[dakuokkonen@fedora ~]\$ mkdir ~/work/arch-pc/lab08" 2. "[dakuokkonen@fedora ~]\$ cd ~/work/arch-pc/lab08" 3. "[dakuokkonen@fedora lab08]\$ touch lab8-1.asm" 4. "[dakuokkonen@fedora lab08]\$".

```
dakuokkonen@fedora:~/work/arch-pc/lab08
[dakuokkonen@fedora ~]$ mkdir ~/work/arch-pc/lab08
[dakuokkonen@fedora ~]$ cd ~/work/arch-pc/lab08
[dakuokkonen@fedora lab08]$ touch lab8-1.asm
[dakuokkonen@fedora lab08]$
```

Рис. 4.1: Работа с директориями и создание файла

Открываю созданный файл `lab8-1.asm`, вставляю в него следующую программу: (рис. 4.2).

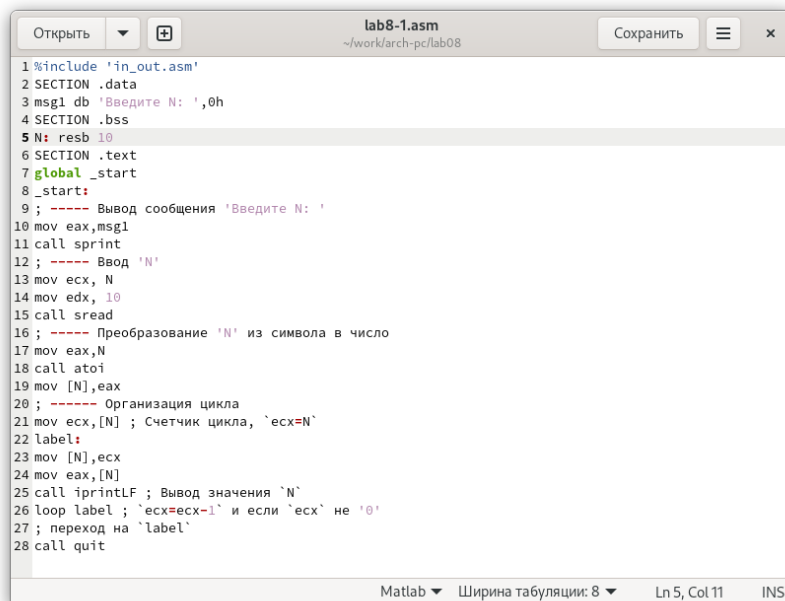


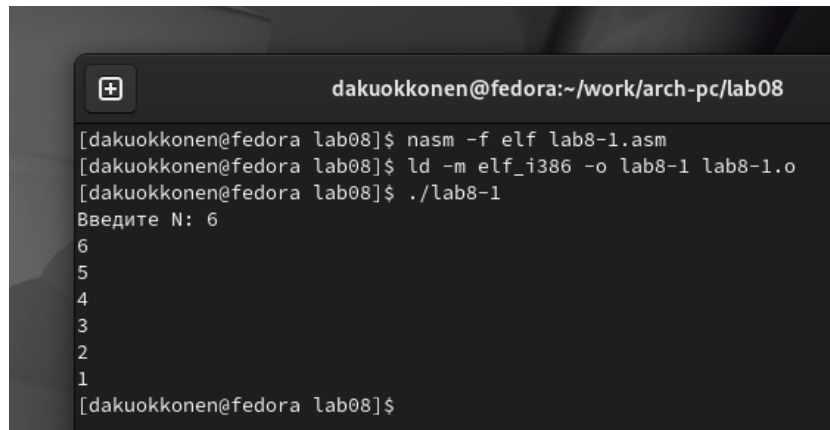
Рис. 4.2: Редактирование файла

Копирую в текущий каталог файл `in_out.asm` с помощью утилиты `cp`, так как он будет использоваться в дальнейшем. (рис. 4.3).

```
[dakuokkonen@fedora lab08]$ cp ~/Загрузки/in_out.asm in_out.asm
```

Рис. 4.3: Копирование файла

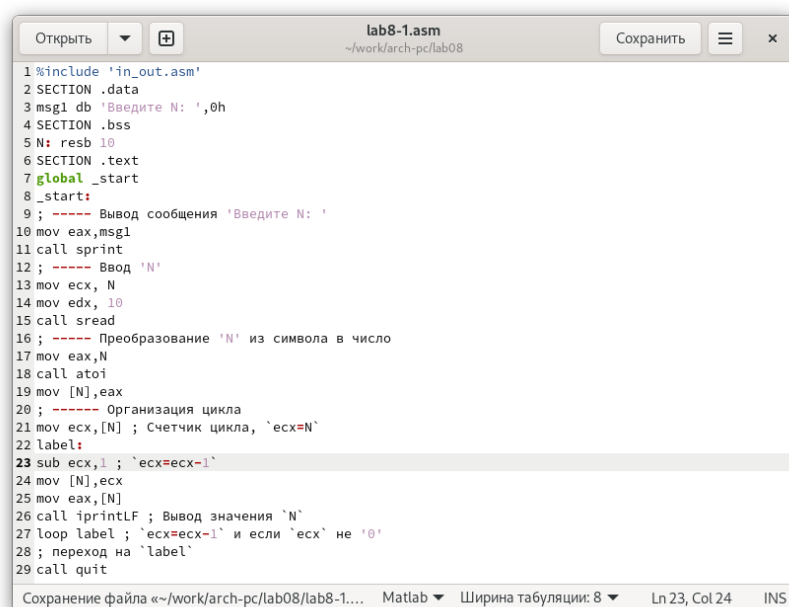
Создаю исполняемый файл и запускаю его. Мы видим, что использование инструкции `loop` позволяет выводить значения регистра `ecx` циклично. (рис. 4.4).



```
dakuokkonen@fedora:~/work/arch-pc/lab08
[dakuokkonen@fedora lab08]$ nasm -f elf lab8-1.asm
[dakuokkonen@fedora lab08]$ ld -m elf_i386 -o lab8-1 lab8-1.o
[dakuokkonen@fedora lab08]$ ./lab8-1
Введите N: 6
6
5
4
3
2
1
[dakuokkonen@fedora lab08]$
```

Рис. 4.4: Подготовка и исполнение файла

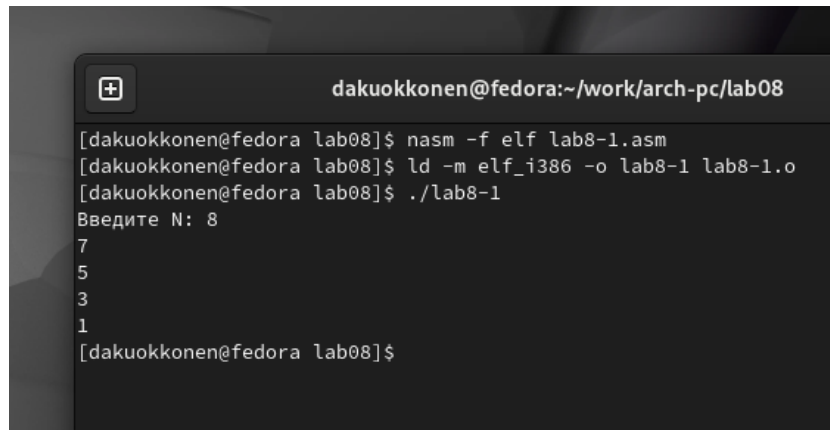
Изменяю значение `ecx` в цикле. (рис. 4.5).



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

Рис. 4.5: Редактирование файла

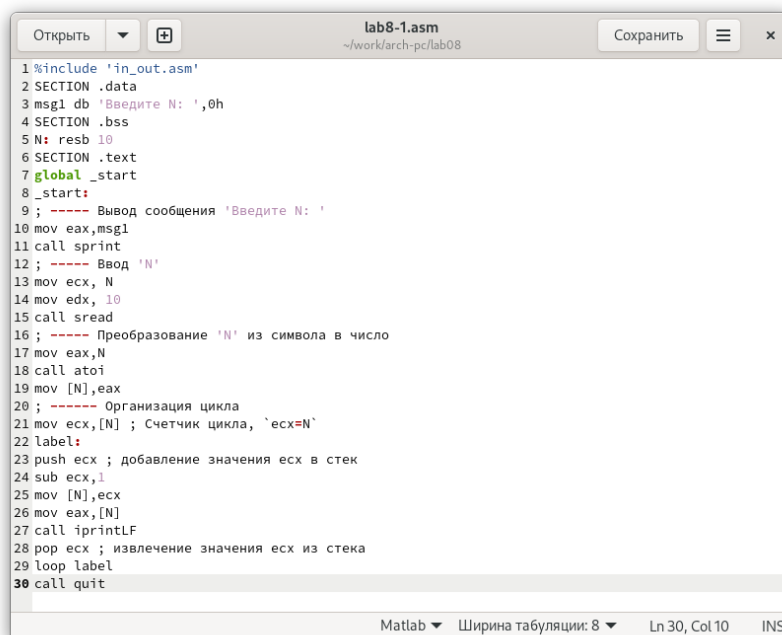
Создаю новый исполняемый файл программы и запускаю его. Мы видим, что регистр `ecx` в цикле принимает совершенно разные значения. И число проходов цикла далеко не соответствует ли значению ☒, введенному с клавиатуры. (рис. 4.6).



```
dakuokkonen@fedora:~/work/arch-pc/lab08
[dakuokkonen@fedora lab08]$ nasm -f elf lab8-1.asm
[dakuokkonen@fedora lab08]$ ld -m elf_i386 -o lab8-1 lab8-1.o
[dakuokkonen@fedora lab08]$ ./lab8-1
Введите N: 8
7
5
3
1
[dakuokkonen@fedora lab08]$
```

Рис. 4.6: Подготовка и запуск исполняемого файла

Вношу изменения в текст программы, добавив команды `push`, для сохранения значения счётчика цикла `loop`. (рис. 4.7).



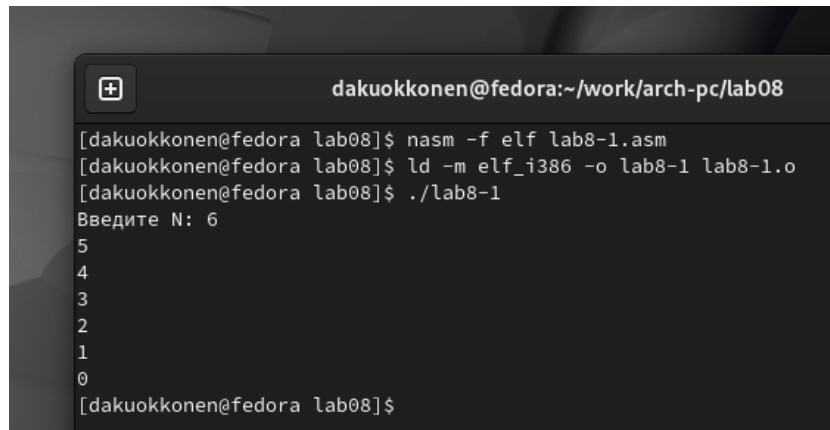
```
lab8-1.asm
~/.work/arch-pc/lab08
Сохранить

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 call quit

Matlab Ширина табуляции: 8 Ln 30, Col 10 INS
```

Рис. 4.7: Создание, редактирование файла

Выполняю компиляцию и компоновку, и запускаю исполняемый файл. В данном случае число проходов цикла соответствует значению ☒ введенному с клавиатуры. Счёт идёт, не от 6-ми, а от 5-ти, но включается 0 (рис. 4.8).

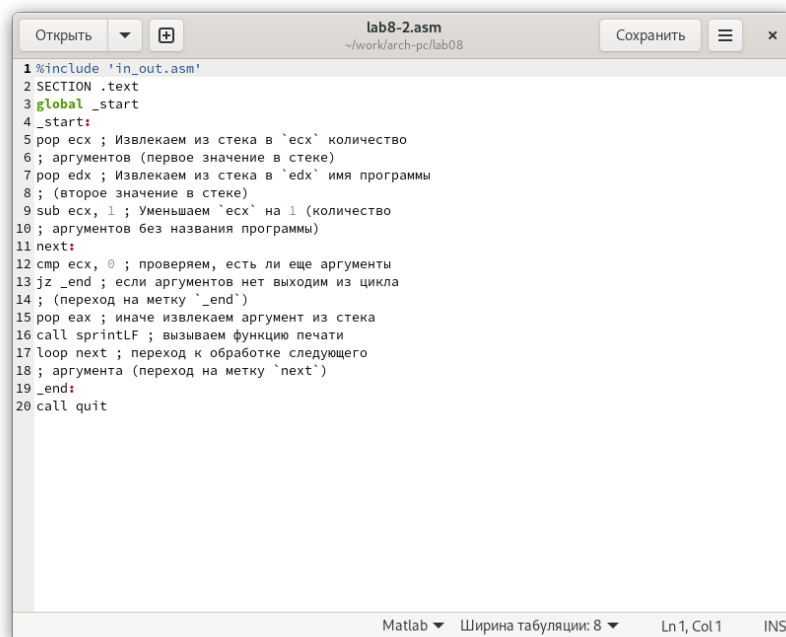


```
dakuokkonen@fedora:~/work/arch-pc/lab08
[dakuokkonen@fedora lab08]$ nasm -f elf lab8-1.asm
[dakuokkonen@fedora lab08]$ ld -m elf_i386 -o lab8-1 lab8-1.o
[dakuokkonen@fedora lab08]$ ./lab8-1
Введите N: 6
5
4
3
2
1
0
[dakuokkonen@fedora lab08]$
```

Рис. 4.8: Создание и запуск исполняемого файла

4.2) Обработка аргументов командной строки.

Создаю файл lab8-2.asm. Редактирую его, вводя предлагаемую программу. (рис. 4.9).



```
lab8-2.asm
~/work/arch-pc/lab08

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. 4.9: Редактирование файла

Создаю исполняемый файл после редактирования. (рис. 4.10).

```
[dakuokkonen@fedora lab08]$ touch lab8-2.asm
[dakuokkonen@fedora lab08]$ gedit lab8-2.asm
[dakuokkonen@fedora lab08]$ nasm -f elf lab8-2.asm
[dakuokkonen@fedora lab08]$ ld -m elf_i386 -o lab8-2 lab8-2.o
```

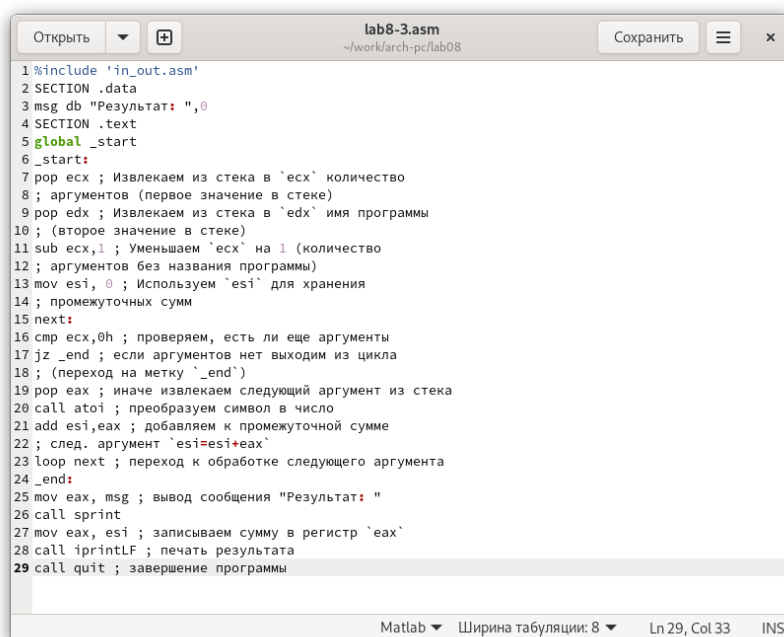
Рис. 4.10: Создание файла, открытие его в режиме правки, компиляция и обработка исполняемого файла

Запускаю исполняемый файл. Программой было обработано 3 аргумента - ровно те, которые я указала при запуске. (рис. 4.11).

```
[dakuokkonen@fedora lab08]$ ./lab8-2 7 9 '2'
7
9
2
[dakuokkonen@fedora lab08]$
```

Рис. 4.11: Запуск исполняемого файла

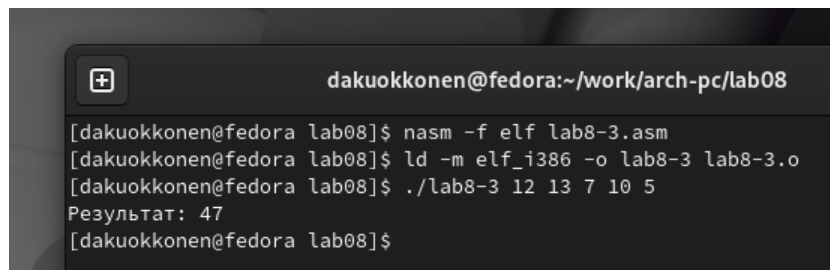
Создаю файл lab8-3.asm. Ввожу в него следующую программу: (рис. 4.12).



```
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. 4.12: Редактирование файла

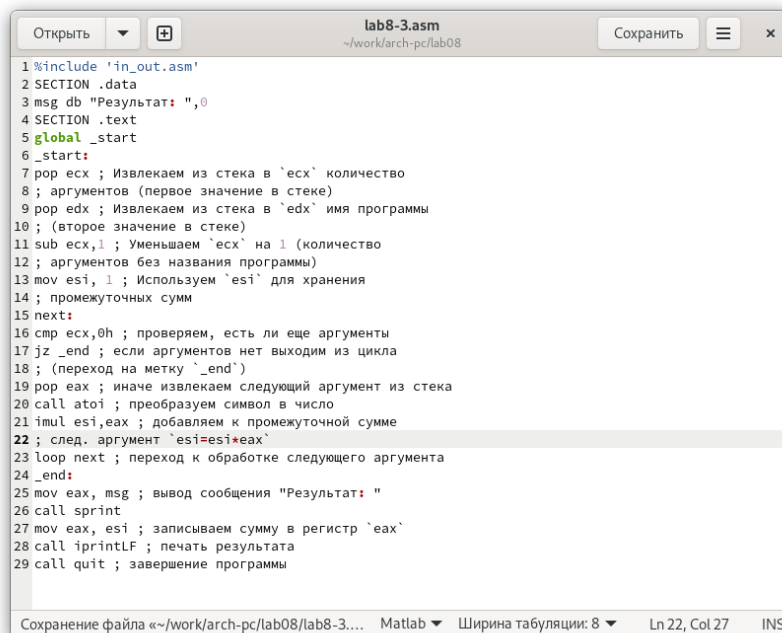
Создаю исполняемый файл. (рис. 4.13).



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

Рис. 4.13: Компиляция и обработка исполняемого файла

Изменяю текст программы для вычисления произведения аргументов командной строки. (рис. 4.14).



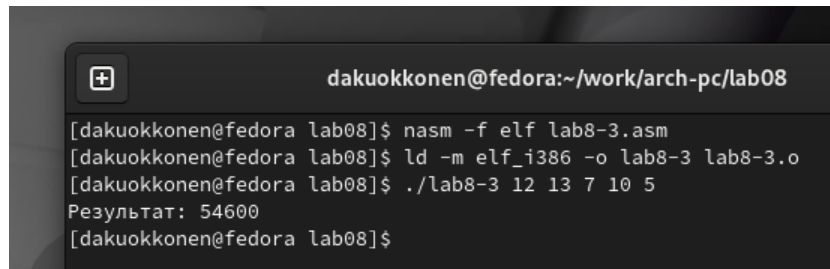
```
lab8-3.asm
~/work/arch-pc/lab08
Сохранить

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

Сохранение файла «~/work/arch-pc/lab08/lab8-3... Matlab Ширина табуляции: 8 Ln 22, Col 27 INS
```

Рис. 4.14: Открытие листинга, редактирование файла

Запускаю исполняемый файл. При проверке видим, что выводятся верные значения. (рис. 4.15).

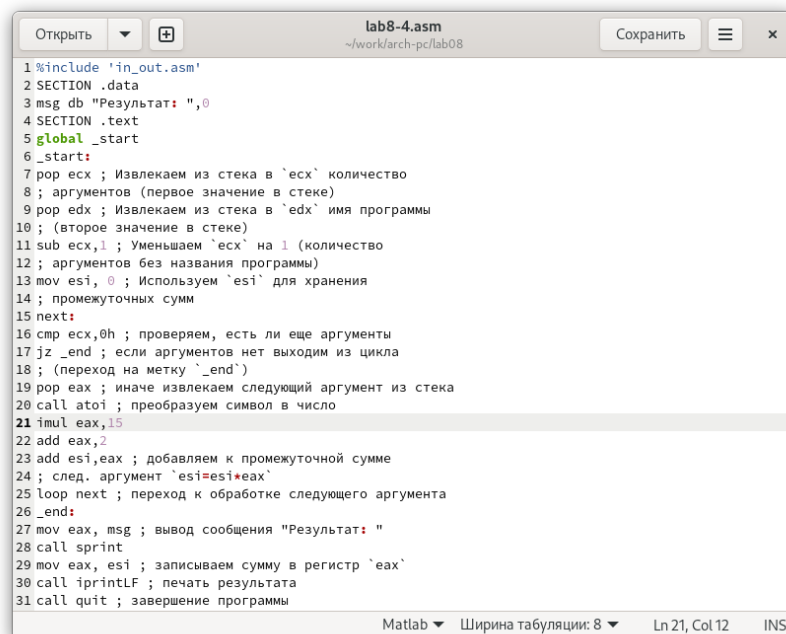


```
dakuokkonen@fedora:~/work/arch-pc/lab08
[dakuokkonen@fedora lab08]$ nasm -f elf lab8-3.asm
[dakuokkonen@fedora lab08]$ ld -m elf_i386 -o lab8-3 lab8-3.o
[dakuokkonen@fedora lab08]$ ./lab8-3 12 13 7 10 5
Результат: 54600
[dakuokkonen@fedora lab08]$
```

Рис. 4.15: Компиляция и запуск исполняемого файла

4.3) Выполнение заданий для самостоятельной работы.

Создаю файл lab8-4.asm с помощью утилиты touch. Открываю созданный файл для редактирования, ввожу в него текст программы для суммирования значений функции, предложенной в варианте 11, полученным мною при выполнении лабораторной работы №6 (рис. 4.16)

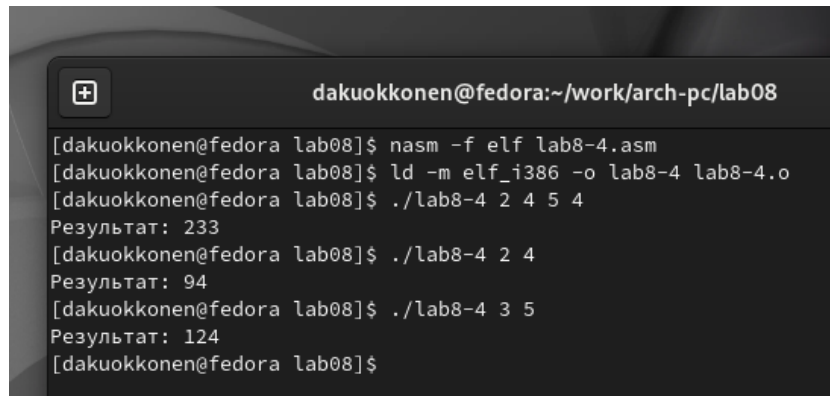


```
lab8-4.asm
~/work/arch-pc/lab08

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 imul eax,15
22 add eax,2
23 add 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
```

Рис. 4.16: Открытие и редактирование файла

Проводим привычные операции и запускаем исполняемый файл, выполняем устную проверку и убеждаемся в правильности работы программы.(рис. 4.17)

A terminal window with a dark background. The title bar shows a plus icon and the text 'dakuokkonen@fedora:~/work/arch-pc/lab08'. The terminal contains the following text:

```
[dakuokkonen@fedora lab08]$ nasm -f elf lab8-4.asm
[dakuokkonen@fedora lab08]$ ld -m elf_i386 -o lab8-4 lab8-4.o
[dakuokkonen@fedora lab08]$ ./lab8-4 2 4 5 4
Результат: 233
[dakuokkonen@fedora lab08]$ ./lab8-4 2 4
Результат: 94
[dakuokkonen@fedora lab08]$ ./lab8-4 3 5
Результат: 124
[dakuokkonen@fedora lab08]$
```

Рис. 4.17: Компиляция, обработка и запуск исполняемого файла

Листинг 4.1 - Программа для суммирования нескольких значений функции, предложенной в варианте 11.

```
%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
add eax,2
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: "
call sprint
mov eax, esi ; записываем сумму в регистр `eax`
call iprintLF ; печать результата
call quit ; завершение программы

```

5 Выводы

При выполнении данной лабораторной работы я приобрела практический опыт в написании программ с использованием циклов и обработкой аргументов командной строки.

Список литературы

Архитектура компьютера и ЭВМ